RELAZIONE

Tabella dei contenuti

1. Considerazioni

Prima di tutto bisogna fare delle considerazioni sull'hardware usato e sul dataset.

1.1. Dataset

Il dataset fornito contiene informazioni sull'andamento giornaliero delle azioni sulla borsa del NYSE e NASDAQ.
Tale dataset è composto da due file csv. Historical_stock_prices.csv contenente circa ventuno milioni di righe, dove ogni riga riporta informazioni sul prezzo di chiusura ed apertura, prezzo minimo, massimo, volume e data di un'azione.
Il secondo file, historical_stocks.csv, contenente poco più di seimila righe, dove ogni riga riporta informazioni sul nome dell'azienda, il settore, l'industria, etc..
Prima di eseguire i vari job viene eseguita una pulizia dei dati mostrata nel notebook. Il dataset sarà quindi formato dai dataframe ripuliti e per effettuare il confronto di come si comportano le implementazioni al crescere della dimensione dell'input, il primo file csv viene suddiviso in vari file, uno da un milione di righe rinominato 1M.csv, uno da tre milioni, 3M.csv ed uno da dieci milioni, 10M.csv.

1.2. Hardware

L'hardware utilizzato in modalità standalone è una macchina virtuale m5.xlarge di Amazon AWS. In modalità cluster invece sono state utilizzate tre macchine m5.xlarge. Tale macchina ha le seguenti specifiche :

Processore	4vCPUs
Memoria	16 GB

2. Pseudocodice

Qui di seguito viene riportato lo pseudocodice per le implementazioni in MapReduce e Spark. Da notare che per MapReduce lo pseudocodice è relativo ad un solo mapper ed un solo reducer per tutti i job, ma nell'implementazione per motivi di efficienza non sempre si è usato un solo mapper ed un solo reducer. Ma per semplicità di lettura si riportano di seguito un solo mapper ed un solo reducer per ogni job.

3. MapReduce

Primo Job :

L'idea è che il mapper filtrerà tutti i campi necessari per eseguire il job. Il reducer invece, si crearà un dizionario per ogni campo di interesse con chiave il ticker e valore una lista (ad esempio, il dizionario relativo al campo "close" avrà come chiave il ticker e valore la lista dei close associati a tale ticker). Così facendo è semplice recuperare la data della prima quotazione, dell'ultima quotazione, del prezzo massimo e minimo del $\text{ticker}_{\text{i-esimo}}$. I rispettivi valori di close corrispondenti alle date sono stati presi attraverso l'indice delle liste. Infine si è calcolata la variazione percentuale.

Primo Job: MapReduce

Mapper

```
Require: standardIO, posizioniCampi;
foreach line in standardInput do
    if line contiene nomi dei campi then
        continue ;
    else
        print(line[posizioniCampi]) ;
                    ▷ stampare quindi line[posizioneTicker],line[posizioneClose],...
end
```

Reducer

```
Require: standardIO, posizioniCampi;
Inizializza un dizionario per ogni campo di interesse, con chiave il nome del ticker e valore
una lista;
procedure CalcolaVariazionePercentuale(xf,xi)
    return ((xf − xi)/xi) ∗ 100
end procedure
foreach line in standardInput do
    d_i[line[ticker]].append(line[nomeCampo_i]);
end
foreach ticker do
    (PrimaData,PrimoClose) = PrendiDataPrimaQuotazioneEdRispettivoClose();
    (UltimaData,UltimoClose) =PrendiDataUltimaQuotazioneEdRispettivoClose();
    variazione = CalcolaVariazionePercentuale(UltimoClose,PrimoClose);
    PrezzoMassimo = PrendiPrezzoMassimo();
    PrezzoMinimo = PrendiPrezzoMinimo();
    print(ticker,PrimaData,UltimaData,variazione,PrezzoMassimo,PrezzoMinimo);
end
```

Secondo Job :

L'idea è che il mapper filtrerà tutti i campi necessari con gli eventuali vincoli per eseguire il job. Il reducer si crea vari dizionari, uno, d_1, con una chiave composta dal ticker e dalla data e come valore la somma dei prezzi di chiusura, l'altro, d_2, con chiave composta da ticker e anno della data e come valore la somma dei volumi, e l'ultimo, d_3, con chiave il settore e valore un set di ticker. Così facendo attraverso d_3 possiamo ottenere facilmente tutti i ticker appartenenti ad un settore e con d_2 otteniamo il volume di transazioni di ogni ticker di ogni anno. Viene usato anche un set di date che appunto conterrà tutte le date. Ordinando il dizionario d_1 l'output risulterà già ordinato. Per ogni settore e per ogni anno di interesse, si ottengono per prima cosa la prima ed ultima data disponibile di quell'anno nel dataset, facilmente ottenibile filtrando dal set di date solo le date con l'anno preso in considerazione e prendere la prima e ultima data. Per ogni ticker del settore preso in considerazione e queste date si ottengono i rispettivi prezzi di chiusura, calcolando così la variazione percentuale di ogni ticker in ogni anno e la variazione percentuale di ogni settore in ogni anno.

Secondo Job: MapReduce

Mapper

```
Require: standardIO, posizioniCampiHSP;
                                    ▷ posizioneCampiHSP = posizioneTicker,posizioneClose,...

Require: posizioniCampiHS;
                                    ▷ posizioneCampiHS =posizioneTicker,posizioneSector,...
procedure FILTRAANNO(anno)
    if 2009 <= anno <= 2018 then
        return True;
    else
        return False;
end procedure

foreach line in standardInput do
    if line contiene nomi dei campi and not FiltraAnno(line[anno]) then
        continue ;
    if line ∈ hsp then
        print(line[posizioniCampiHSP]) ;
                    ▷ stampare quindi line[posizioneTicker],line[posizioneClose],...
    else
        print(line[posizioniCampiHS]) ;
                    ▷ stampare quindi line[posizioneTicker],line[posizioneSector],...
end
```

Reducer

```
Require: standardIO, posizioniCampi;
Inizializza un dizionario con chiave composta da ticker e data e valore il rispettivo prezzo
  di chiusura ,d_C
Inizializza un dizionario con chiave composta da ticker e anno della data e valore la
  somma dei volumi di quel ticker in quell'anno,d_V ;
Inizializza un dizionario con chiave il settore e valore un insieme di ticker,d_T ;
Inizializza un insieme che conterrà le date, dates;
procedure CALCOLAVARIAZIONEPERCENTUALE(xf,xi)
    return ((xf − xi)/xi) ∗ 100;
end procedure
foreach line in standardInput do
    if line ∈ hsp then
        d_C[line[ticker]+line[data]] += line[close];
        d_V[line[ticker]+line[data].year] += line[volume];
        dates.add(line[data]);
    else
        d_T[line[settore_j]].add(line[ticker]);
end
Ordina il dizionario d_T ;
                                    ▷ In base alla chiave (nome del settore)
```

```
foreach sector in d_T do
    foreach anno di interesse 'a' do
        dataMin,dataMax = getDate('a');
        xiSector = 0 ;
        xfSector = 0 ;
        maxTickerVariation = float('-inf') ;
        tickerVariation = 0 ;
        tickerVolume = 0 ;
        maxVolume = 0;
        foreach ticker ∈ sector do
            xi = getClose(dataMin);
            xf = getClose(dataMax);
            xiSector += xi;
            xfSector += xf;
            var = CalcolaVariazionePercentuale(xf,xi);
            if var > maxTickerVariation then
                maxTickerVariation = var ;
                tickerVariation = ticker;
            tmp = getMaxVol(ticker,'a');
            if tmp > maxVolume then
                maxVolume=tmp;
                tickerVolume = ticker;
        end
    varSettore = CalcolaVariazionePercentuale(xfSector,xiSector);
    print(ticker,'a',varSettore,tickerVariation,maxTickerVariation,tickerVolume,maxVolume);
end
```

Terzo Job :

L'idea è che il mapper filtrerà tutti i campi necessari con gli eventuali vincoli per eseguire il job. Similmente al secondo job il reducer si crea i dizionari necessari, uno, d_1, con chiave composta da data e ticker e come valore il prezzo di chiusura e l'altro, d_2, con chiave il mese e valore un set di date in quel mese, e infine si crea un set di tickers. Per ogni mese si prendono la prima ed ultima data di quel mese e per ogni ticker si otterranno i prezzi di chiusura delle rispettive date per poi calcolare la variazione percentuale mensile. Questo risultato verrà aggiunto ad un dizionario (d_3) con chiave il ticker e valore la lista delle variazioni. Si procede poi all'estrazione di tutte le coppie "simili" in d_3; esse sono simili se il valore assoluto della differenza di tutte variazioni percentuali (rispettive agli stessi mesi, ovvero il primo ticker del mese di gennaio con il secondo di gennaio, stessa cosa per febbraio, e così via per gli altri mesi) non supera una determinata soglia.

Terzo Job: MapReduce

Mapper

```
Require: standardIO, posizioniCampi;
foreach line in standardInput do
    if line[data].year == 2017 then
        print(line[posizioniCampi]) ;
                    ▷ stampare quindi line[posizioneTicker],line[posizioneClose],...
end
```

Reducer

```
Require: standardIO, posizioniCampi;
Inizializza un dizionario con chiave composta da data e ticker e valore il prezzo di
chiusura,d_DT
Inizializza un dizionario con chiave il ticker e valore la lista delle variazioni percentuali
mensili,d_T
Inizializza un dizionario con chiave la il mese della data e valore un insieme delle date in quel
mese,d_D;
Inizializza un insieme che conterrà i tickers, tickers;
Imposta costante SOGLIA = 1;
procedure CalcolaVariazionePercentuale(xf,xi)
    return ((xf − xi)/xi) ∗ 100;
end procedure

procedure checkIfSimilar(l1,l2) flag = True;
    for i ← 0 to len(l1) do
        if |l1[i] − l2[i]| <= SOGLIA then
            flag = True;
        else
            return False;
    end
    return flag;
end procedure
foreach line in standardInput do
    d_DT[line[data]+line[ticker]] = line[close];
    d_D[line[data].month].add(line[data]);
    tickers.add(line[ticker]);
end
for mese ← 1 to 13 do
    dataMin = min(d_D.get(mese));
    dataMax = max(d_D.get(mese));
    foreach ticker in tickers do
        xi = d_DT.get(dataMin+ticker);
        xf = d_DT.get(dataMax+ticker);
        variazionePercentuale = CalcolaVariazionePercentuale(xf,xi);
        d_T[ticker].append(variazionePercentuale)
    end
end
foreach elem in d_T do
    controlla se è simile ad un altro elemento in d_T,
    se si stampa i due ticker e le relative liste
end
```

4. Spark

Funzioni generali :

GetMin

```
procedure GETMIN(x,y)
    if x[1] > y[1] then
        return y;
          else
        return x;
end procedure
```

GetMax

```
procedure GETMAX(x,y)
    if x[1] > y[1] then
        return x;
          else
        return y;
end procedure
```

Primo Job :

L'idea è che dopo aver filtrato i campi necessari al primo job, essi verranno salvati in un main rdd. Si estraggono degli rdd più piccoli per poi unirli per eseguire il job. In particolare, si creano degli rdd aventi come chiave il ticker e valore la data, per ottenere poi la data della prima e ultima quotazione di ogni ticker. In modo analogo si procede per estrarre il prezzo massimo e minimo. I rispettivi prezzi di chiusura delle date sopra citate si ottengono creando rdd con chiave ticker e valore una tupla formata dal prezzo di chiusura e dalla data. Si estraggono poi i prezzi relativi alle date precedenti e si calcolerà la variazione percentuale. Verrà infine eseguito un join per completare il job.

PrimoJob

```
inputFilePath;
outputFilePath;
spark = createSparkSession();
inputRDD = spark.read(inputFilePath).cache().filter(firstLine);
            ▷ La prima riga non viene considerata, contiene i nomi dei campi;
mainTable = inputRDD.map(ticker,close,low,high,data);
          ▷ mappata cosi da avere rdd formati (ticker, close, low, high, data);
mainTable.persist(StorageLevel.MEMORY AND DISK);
tickerMinDate = mainTable.map((ticker,data)).reduceByKey(min);
tickerMaxDate = mainTable.map((ticker,data)).reduceByKey(max);
firstDataClose = mainTable.map((ticker,(close,data))).reduceByKey(GetMin(x, y));
            ▷ mappata cosi da avere rdd formati (ticker, (close, firstdata));
lastDataClose = mainTable.map((ticker,(close,data))).reduceByKey(GetMax(x, y));
            ▷ mappata cosi da avere rdd formati (ticker, (close, lastdata));
tickerPriceMin = mainTable.map((ticker,low)).reduceByKey(min);
tickerPriceMax = mainTable.map((ticker,high)).reduceByKey(max);
tickerClose = firstDataClose.join(lastDataClose);
          ▷ mappata cosi da avere rdd formati (ticker, (variazionepercentuale));
result = tickerMinDate.join(tickerMaxDate).join(tickerClose)
          .join(tickerPriceMin).join(tickerPriceMax);
                                    ▷ eseguire adeguato mapping;
out = result.sortBy(campo relativo a lastDataClose, ascending=False);
          ▷ ordiniamo in modo decrescente per data dell'ultima quotazione;
out.coalesce(1, shuffle = True).saveAsTextFile(outputFilePath)
```

Secondo Job :

L'idea è che dopo aver filtrato i campi necessari al secondo job, essi verranno salvati in un main rdd. Per ottenere il ticker che ha avuto il maggior numero di transazioni per ogni anno di interesse, si crea un rdd con chiave una tripla composta da settore, anno e ticker, e come valore il volume. Attraverso la funzione reduceByKey e adeguati mapping calcoliamo la somma dei volumi di ogni ticker per ogni anno. Per ottenere la variazione percentuale annuale di ogni settore di ogni anno, viene costruito un rdd con chiave la coppia settore, data e valore il prezzo di chiusura. Poi attraverso la funzione reduceByKey e opportuni mapping si ottiene un rdd con chiave la coppia settore, anno e come valore la somma di tutti i prezzi di chiusura relativi all'ultima data disponibile del relativo anno, analogo per la prima data. Questi ultimi due rdd verranno uniti e verrà calcolata la variazione percentuale di ogni anno di ogni settore. In modo simile verranno estratti per ogni settore e per ogni anno il ticker con la variazione percentuale maggiore. Il risultato sarà l'unione di questi rdd.

SecondoJob

```
inputFilePathHSP;
inputFilePathHS;
outputFilePath;
spark = createSparkSession();
inputRDDHSP = spark.read(inputFilePathHSP).cache().filter(firstLine and 2009 <=
  data.year <= 2018 );
    ▷ La prima riga non viene considerata, consideriamo solo gli anni che vanno dal 2009 al
  2018;
inputRDDHS = spark.read(inputFilePathHS).cache().filter(firstLine and nan);
                    ▷ La prima riga e i valori nulli non vengono considerati;
mainTable = inputRDDHSP.join(inputRDDHS);
        ▷ Opportuno mapping cosi da avere rdd (ticker, close, volume, date, settore);
mainTable.persist(StorageLevel.MEMORY AND DISK);
maxVolume = mainTable.map(((sector,year,ticker),volume)) usando reduceByKey e
  adeguati mapping calcoliamo la somma dei volumi per ogni ticker e prenderemo per ogni
  anno il ticker con la somma dei volumi massima e il relativo valore;
       ▷ Il risultato sarà rdd formati ((sector, year), tickersommavolumemassima);
sectorCloseYearDataMax = mainTable.map(((sector,data),close))
  .reduceByKey(add).map(((sector,data.year),(close,data))).reduceByKey(GetMax(x, y));
            ▷ Con opportuno mapping ((sector, year), sum close of date max);
In modo analogo calcoliamo sectorCloseYearDataMin;
sectorVariationYear = sectorCloseYearDataMax.join(sectorCloseYearDataMin);
          ▷ Con opportuno mapping calcoliamo la variazione percentuale dell'anno;
sectorTickerCloseDataMax =
  mainTable.map(((sector,data.year,ticker),(close,data))).reduceByKey(GetMax(x, y)) ;
                                  ▷ ((sector, year), ticker, var);
Analogo per la data minima avremo sectorTickerCloseDataMin;
sectorTickerVar =
  sectorTickerCloseDataMax.join(sectorTickerCloseDataMin).map(calcolo variazione
  percentuale).reduceByKey(GetMax(x, y)) ;
            ▷ Cosi abbiamo per ogni settore e per ogni anno il ticker con maggiore variazione
  percentuale;
result = sectorVariationYear.join(sectorTickerVar).join(maxVolume).sortBy(settore);
result.coalesce(1,shuffle=True).saveAsTextFile(outputFilePath)
```

Terzo Job :

L'idea è che dopo aver filtrato i campi necessari al terzo job, essi verranno salvati in un main rdd. Il procedimento è simile a quello del secondo job, ma invece di usare l'anno si userà il mese, ottenendo così due rdd. Attraverso la loro unione e opportuni mapping si calcolerà la variazione percentuale mensile di ogni mese per ogni job. In seguito si esegue un prodotto cartesiano e si estraggono le coppie che sono simili; esse sono simili se il valore assoluto della differenza di tutte le variazioni percentuali (rispettive agli stessi mesi, ovvero il primo ticker del mese di gennaio con il secondo di gennaio, stessa cosa per febbraio, e così via per gli altri mesi) non supera una determinata soglia.

Terzo Job

SOGLIA = 1;
inputFilePathHSP;
outputFilePath;
spark = createSparkSession();
inputRDDHSP = spark.read(inputFilePathHSP).cache().filter(firstLine and data.year == 2017);

▷ La prima riga non viene considerata, consideriamo solo l'anno 2017;

mainTable = inputRDDHSP.map((ticker,close,data));
mainTable.persist(StorageLevel.MEMORY AND DISK);
tickerMonthCloseDataMax = mainTable.map(((ticker, data.month), (close,data))).reduceByKey(GetMax(x, y));

▷ Avremo RDD formati $((ticker, mese)close)$ cioè per ogni ticker e per ogni mese il prezzo di chiusura della data dell'ultimo mese;

Analogalmente otterremo tickerMonthCloseDataMin;
tickerMonthVar = tickerMonthCloseDataMax.join(tickerMonthCloseDataMin) con opportuni mapping e raggruppamenti calcoleremo la variazione percentuale mensile che dovrà diventare il valore di una colonna;
prodCart = tickerMonthVar.cartesian(tickerMonthVar);

▷ Filtrano però i ticker con lo stesso nome;

Con un opportuno mapping andiamo a calcolare il valore assoluto della sottrazione delle variazione percentuali mensili delle varie coppie di ticker che se inferiore di SOGLIA allora farà parte del risultato;
result.coalesce(1,shuffle=True).saveAsTextFile(outputFilePath)

5. Risultati

Qui di seguito vengono riportati gli output dei vari job.

Primo Job : In ordine da sinistra verso destra, ticker, data della prima quotazione, data dell'ultima quotazione, variazione percentuale della quotazione, prezzo massimo e prezzo minimo. Ordinati in modo decrescente per data dell'ultima quotazione e per nome del ticker

TICKER	DATA PRIMA QUOTAZIONE	DATA ULTIMA QUOTAZIONE	VARIAZIONE PERCENTUALE	PREZZO MINIMO	PREZZO MASSIMO
A	1999-11-18	2018-08-24	109.636%	10.000	99.383
AA	1970-01-02	2018-08-24	508.325%	10.008	99.893
AABA	1996-04-12	2018-08-24	4910.909%	0.646	98.500
AAC	2018-01-16	2018-08-24	4.857%	10.000	9.975
AAL	2005-09-27	2018-08-24	101.140%	1.450	9.990
AAME	1980-03-17	2018-08-24	-29.870%	0.375	9.900
AAN	1987-01-20	2018-08-24	4683.263%	0.481	9.956
AAOI	2013-09-26	2018-08-24	330.422%	10.000	99.980
AAON	1992-12-16	2018-08-24	41348.204%	0.090	9.996
AAP	2001-11-29	2018-08-24	1084.150%	100.020	99.990

Secondo Job : In ordine da sinistra verso destra, settore, anno, l'azione con maggiore variazione percentuale e tale valore, l'azione del settore con maggior volume di transazione e tale valore. Ordinati per nome del settore e anno.

SETTORE	ANNO	VARIAZIONE PERCENTUALE	AZIONE	VARIAZIONE PERCENTUALE	AZIONE	VOLUME
BASIC INDUSTRIES	2009	36.016%	GURE	709.722%	FCX	9141685400.0
BASIC INDUSTRIES	2010	28.566%	WWR	319.753%	FCX	6891808600.0
BASIC INDUSTRIES	2011	-24.465%	VHI	188.640%	FCX	5150807800.0
BASIC INDUSTRIES	2012	2.354%	PATK	261.860%	VALE	4659766700.0
BASIC INDUSTRIES	2013	201.537%	XRM	416.928%	VALE	4428233700.0
BASIC INDUSTRIES	2014	-82.931%	CENX	134.841%	VALE	5660183200.0
BASIC INDUSTRIES	2015	-2.046%	AMWD	100.050%	FCX	7286761300.0
BASIC INDUSTRIES	2016	20.355%	TECK	451.791%	FCX	10464699500.0
BASIC INDUSTRIES	2017	16.811%	OPNT	310.179%	VALE	7023267600.0
BASIC INDUSTRIES	2018	-3.470%	XRM	213.817%	VALE	3710091900.0

Terzo Job : In ordine da sinistra verso destra, la coppia di ticker che si somigliano e le loro variazione percentuali da gennaio a dicembre.

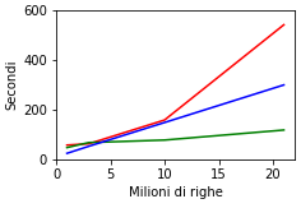
```
TICKER1,TICKER2 GEN     FEB     MAR     APR     MAG     GIU     LUG     AGO     SET     OTT     NOV     DIC
AAXJ,AIA        [('6.346', '6.454'), ('2.589', '1.926'), ('2.843', '2.619'), ('1.278', '1.205'), ('3.632', '3.929'), ('0.104', '0.811'), ('4.746', '5.208'), ('1.151', '0.544'), ('-0.290',
'0.393'), ('3.905', '4.898'), ('-0.434', '0.229'), ('1.531', '1.413')]
AAXJ,EEMA       [('6.346', '6.456'), ('2.589', '2.333'), ('2.843', '2.634'), ('1.278', '1.457'), ('3.632', '4.088'), ('0.104', '0.746'), ('4.746', '4.905'), ('1.151', '1.121'), ('-0.290',
'-0.172'), ('3.905', '4.516'), ('-0.434', '-0.544'), ('1.531', '1.430')]
ACWX,IXUS       [('3.260', '3.485'), ('0.738', '1.004'), ('1.868', '1.747'), ('2.085', '2.058'), ('2.660', '2.381'), ('-1.788', '-1.376'), ('3.532', '3.466'), ('-0.063', '0.117'), ('1.661',
'1.566'), ('2.008', '1.984'), ('0.142', '0.321'), ('1.134', '1.106')]
ACWX,VXUS       [('3.260', '3.333'), ('0.738', '1.002'), ('1.868', '1.491'), ('2.085', '2.074'), ('2.660', '2.494'), ('-1.788', '-1.179'), ('3.532', '3.288'), ('-0.063', '-0.092'),
('1.661', '1.052'), ('2.008', '2.028'), ('0.142', '0.303'), ('1.134', '1.266')]
AGC,NCV [('3.248', '4.225'), ('3.483', '4.211'), ('-2.707', '-2.734'), ('1.967', '2.655'), ('-0.159', '-0.287'), ('-0.954', '-0.429'), ('2.552', '2.845'), ('-3.577', '-3.444'), ('0.161',
'1.140'), ('-0.801', '-0.559'), ('-1.951', '-1.127'), ('-0.663', '-0.566')]
AIA,EEMA        [('6.454', '6.456'), ('1.926', '2.333'), ('2.619', '2.634'), ('1.205', '1.457'), ('3.929', '4.088'), ('0.811', '0.746'), ('5.208', '4.905'), ('0.544', '1.121'), ('0.393',
'-0.172'), ('4.898', '4.516'), ('0.229', '-0.544'), ('1.413', '1.430')]
BNDX,IEF        [('-0.628', '0.267'), ('0.988', '0.917'), ('0.092', '0.783'), ('0.184', '0.613'), ('0.718', '1.045'), ('-0.403', '-0.458'), ('0.350', '0.745'), ('0.531', '1.206'), ('-
0.347', '-1.151'), ('0.734', '-0.075'), ('0.146', '-0.339'), ('-1.289', '-0.340')]
BNDX,IEI        [('-0.628', '0.318'), ('0.988', '0.392'), ('0.092', '0.556'), ('0.184', '0.398'), ('0.718', '0.510'), ('-0.403', '-0.299'), ('0.350', '0.568'), ('0.531', '0.589'), ('-
0.347', '-0.628'), ('0.734', '-0.130'), ('0.146', '-0.382'), ('-1.289', '-0.294')]
BNDX,VBND       [('-0.628', '0.266'), ('0.988', '0.592'), ('0.092', '0.059'), ('0.184', '0.466'), ('0.718', '0.677'), ('-0.403', '-0.870'), ('0.350', '0.993'), ('0.531', '0.540'), ('-
0.347', '-0.664'), ('0.734', '-0.130'), ('0.146', '-0.575'), ('-1.289', '-1.137')]
BNDX,VGIT       [('-0.628', '0.281'), ('0.988', '0.500'), ('0.092', '0.626'), ('0.184', '0.983'), ('0.718', '0.822'), ('-0.403', '-0.293'), ('0.350', '0.590'), ('0.531', '0.756'), ('-
0.347', '-0.676'), ('0.734', '-0.124'), ('0.146', '-0.295'), ('-1.289', '-0.312')]
```

6. Grafici

Di seguito vengono mostrati i grafici che mettono a confronto i tempi di esecuzione delle varie implementazioni al variare della grandezza del dataset, sia in modalità standalone che in cluster. Nelle tabelle, il valore che vi è nelle celle è in secondi, ogni riga rappresenta una implementazione e le colonne la grandezza (in milioni di righe) del file di input.

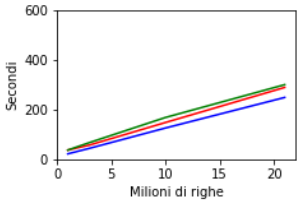
Primo Job Standalone

	1M	3M	10M	21M
MapReduce	60	65	160	540
Hive	50	70	80	120
Spark	27	55	150	300



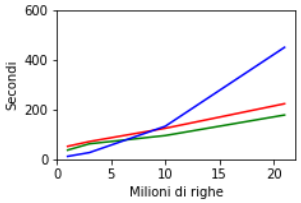
Secondo Job Standalone

	1M	3M	10M	21M
MapReduce	40	60	150	290
Hive	40	70	170	301
Spark	25	47	128	250



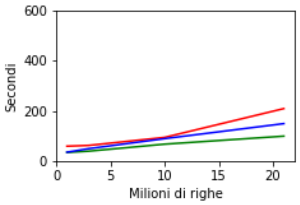
Terzo Job Standalone

	1M	3M	10M	21M
MapReduce	55	74	127	225
Hive	40	65	98	180
Spark	15	30	135	450



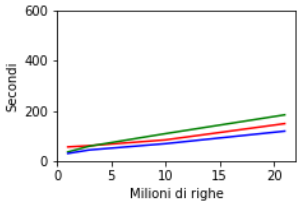
Primo Job Cluster

	1M	3M	10M	21M
MapReduce	60	63	95	210
Hive	35	40	68	100
Spark	36	50	90	150



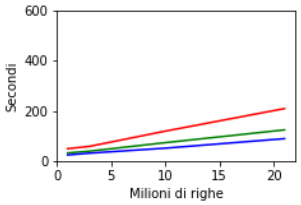
Secondo Job Cluster

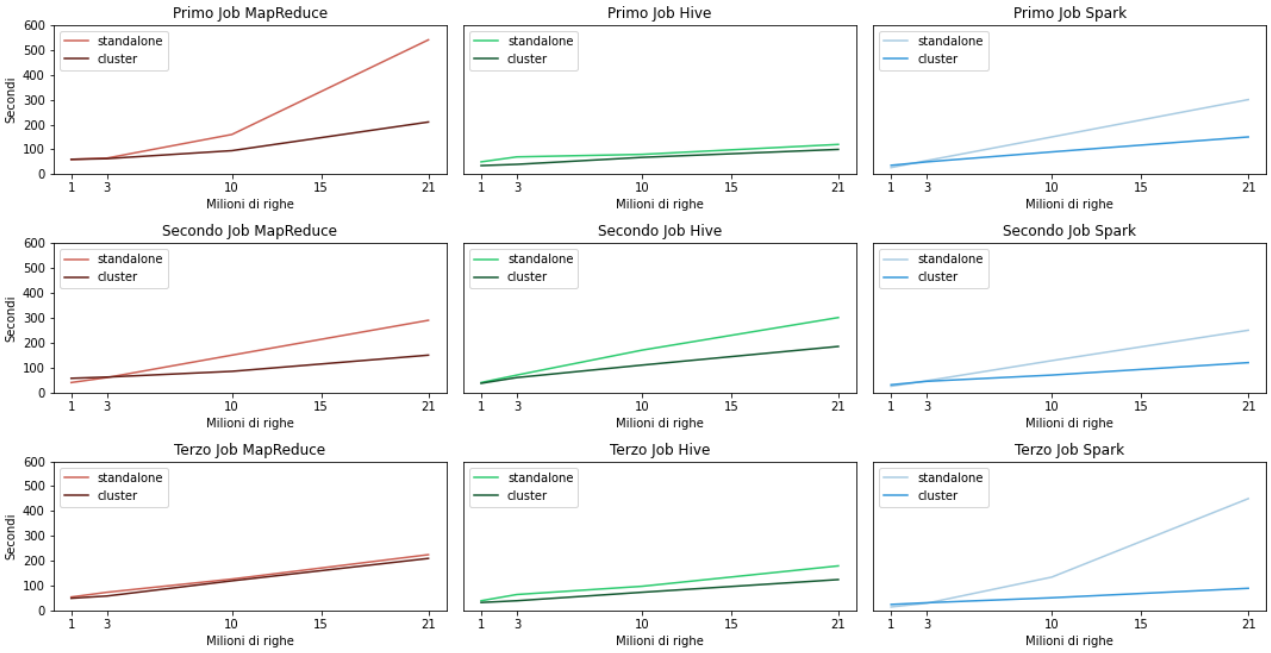
	1M	3M	10M	21M
MapReduce	57	62	85	150
Hive	37	60	110	185
Spark	31	45	70	120



Terzo Job Cluster

	1M	3M	10M	21M
MapReduce	50	59	120	210
Hive	33	40	74	125
Spark	25	32	52	90





7. Implementazione

Tutto il codice delle tre implementazioni (MapReduce, Hive e Spark) e il relativo output è scaricabile dal repository GitHub. Esso è strutturato come segue: nella root del progetto vi è un notebook per effettuare la pulizia del dataset, salvando un nuovo dataframe nella cartella “dataset” dove vi sono anche i dataframe originali. Vi è una cartella per ogni job, ed al suo interno vi è l’output ottenuto e il codice delle tre implementazioni.
In particolare, per l’implementazione in MapReduce si è cercato di ottimizzare il codice per la modalità standalone differenziandolo dal codice per la modalità in cluster, cercando di usare meno mapper e reducer. Le scelte effettuate per i vari job sono le seguenti.
La variazione percentuale si è calcolata tramite l’espressione: $((x_f-x_i)/x_i)*100$, dove x_f rappresenta l’ultimo prezzo di chiusura e x_i il primo prezzo di chiusura. In, particolare per il primo job, per ottenere il prezzo massimo, si è preso il massimo del campo “high”, mentre per ottenere il prezzo minimo si è preso il minimo del campo “low”. Per il secondo job invece per calcolare la variazione percentuale del settore nell’anno, si sono inizialmente ottenute la prima e ultima data presenti nel dataset relative all’anno considerato. Una volta presi tutti i prezzi di chiusura di tutti i ticker di un settore nella data iniziale si sono sommati, stessa cosa è stata fatta per i prezzi di chiusura dell’ultima data. Infine, si è calcolata la variazione percentuale, dove x_i sarà la somma di tutti i prezzi di chiusura della prima data disponibile del dataset nel relativo anno, e x_f la somma dei prezzi di chiusura dell’ultima disponibile del dataset nel relativo anno.
In modo analogo si è ottenuta la variazione percentuale annuale dei ticker. Per semplicità, per il terzo job si è deciso di mostrare le coppie di ticker che si somigliano e non il nome delle relative aziende.
Per facilitare la lettura non viene riportato in questo report le implementazioni in Hive, ma esse sono scaricabili dal repository, in particolare, primoJob, secondoJob, terzoJob.