Primo Progetto Big Data - 22 Maggio 2021

Andrea Giorgi AND.GIORGI4@STUD.UNIROMA3.IT
Dipartimento di Ingegneria Informatica
Università degli studi Roma Tre
Matricola: 508800

Francesco Forgione FRA.FORGIONE1@STUD.UNIROMA3.IT
Dipartimento di Ingegneria Informatica
Università degli studi Roma Tre
Matricola: 502243

Progetto: Analizzare le azioni di mercato sulla borsa di New York (NYSE) e sul NASDAQ

Abstract

Il termine Big Data indica genericamente una raccolta di dati informativi così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valore o conoscenza. Negli ultimi anni con l'avvento di grande moli di dati, causati dal notevole incremento di flusso di dati nel web e dal parallelo incremento delle capacità di elaborazione e memorizzazione dei dati, i Big Data hanno assunto un ruolo fondamentale nella gestione dei dati, soprattutto in relazione alla misurazione delle prestazioni di un'organizzazione nonché di un processo aziendale. Il termine è utilizzato dunque in riferimento alla capacità (propria della scienza dei dati) di analizzare ovvero estrapolare e mettere in relazione un'enorme mole di dati eterogenei, strutturati e non strutturati, al fine di scoprire i legami tra fenomeni diversi. La scienza dei dati la si ritrova in diversi ambiti, dal campo medico e biologico fino al campo dell'areospazio. Mondi così diversi hanno come punto di raccordo proprio il dover saper sfruttare delle importanti moli di dati nella maniera più efficiente possibile, sia in termini di prestazioni che di scalabilità e capacità di memorizzazione.

Keywords: BigData, Hadoop, Hive, Spark, AWS.

1. Introduzione

In questo elaborato verranno descritte le procedure di analisi ed elaborazione dati proprie dei Big Data applicate al campo della finanza. A partire dai dati relativi all'andamento giornaliero di una rosa di azioni sulla borsa di New York (NYSE) e sul NASDAQ dal 1970 al 2018 verranno utilizzate diverse tecnologie per compiere delle operazioni di analisi ed elaborazione. Queste tecnologie sono state implementate e sfruttate in un ambiente virtualizzato basato su sistema operativo Ubuntu. L'utilizzo di un ambiente virtualizzato, tramite l'utilizzo della tecnologia VirtualBox offerta da Oracle, ha permesso di ottimizzare al massimo l'utilizzo delle risorse durante la fase di elaborazione dei dati, andando così a focalizzare la nostra attenzione non soltanto sulla mera parte implementativa ma anche sulla fase di ottimizzazione del codice. Verrà presentata un introduzione per ciascun Job di elaborazione, per poi seguire con una descrizione dettagliata della fase di studio ed implementazione di ciascuna soluzione. Le scelte implementative verranno descritte sia dal

punto di vista tecnico che dal punto di vista prestazionale, mostrandone le prestazione a fronte di un input ridotto e dell'input completo. L'esecuzione di queste soluzioni verrà effettuata anche su una distribuzione cloud basata su soluzioni offerte da AWS, in modo tale da valutare le prestazioni anche in ambiente distribuito e non locale.

2. Analisi del dataset

Il dataset complessivo sul quale verteranno le implementazioni descritte nelle sezioni successive è composto da due file distinti: (1) *Historical stock prices* e (2) *Historical stocks*. Entrambi i file sono stati forniti in formato csv, permettendo di utilizzare le tecniche note di processamento dei file fornite dal linguaggio di programmazione utilizzato, ossia Python 3.x.

2.1 Historical stock prices

Questo dataset contiene al suo interno diversi campi informativi utili per poter descrivere le informazioni salienti di ciascuna azione, ognuna delle quali viene identificata tramite un valore univoco *ticker*. Le dimensioni del dataset sono di media grandezza, nell'ordine dei 2GB di dimensione. Un analisi preliminare mostra come questo file non contenga valori mancanti ne inconsistenze di valori, permettendoci di non pre-elaborarlo. Si può notare come una colonna in particolare, *adj_close*, non sia di interesse per le elaborazioni da eseguire, ma è stato deciso di non andare ad eliminarla per non intaccare la struttura del file csv. Data la mole importante di questo dataset è stata presa la decisione di prelevare una sezione di questo creando un nuovo dataset, chiamato *historical_stocks_prices_debug.csv*, che permette di valutare le prestazione delle soluzioni implementate a fronte di un input minore e di rendere eseguibili queste soluzioni anche su dispositivi con ridotte capacità di elaborazione.

2.2 Historical stocks

Questo dataset contiene al suo interno le informazioni necessarie per identificare a partire dall'identificatore univoco *ticker* l'azienda connessa all'azione e il mercato di riferimento. Le dimensioni di questo file sono irrisorie rispetto al dataset precedente e contiene al suo interno 15 records. Ciascun record permette di associare un set di azioni ad una singola azienda.

3. Job 1

L'obiettivo di questo Job è quello di generare un report contenente per ciascuna azione: (1) la data della prima quotazione, (2) la data dell'ultima quotazione, (3) la variazione percentuale della quotazione, (4) il prezzo massimo e quello minimo e (5) (facoltativo) il massimo numero di giorni consecutivi in cui l'azione è cresciuta con indicazione dell'anno in cui questo è avvenuto. Il report sarà ordinato per valori decrescenti del punto b.

3.1 Implementazione MapReduce

L'implementazione in MapReduce è stata basata sul linguaggio di programmazione Python. Si focalizza sulla creazione di due script, uno script mapper ed uno reducer. Il mapper ef-

fettua un'operazione di filtraggio andando a selezionare i campi di interesse per il suddetto Job, nello specifico *ticker, open, close, lowThe, highThe, date* e ne effettua una validazione, ossia controlla che nessuno di questi campi presenti un valore nullo al momento della lettura. Lo script reducer invece effettua le operazioni necessarie per il raggiungimento degli obiettivi presentati in precedenza. Osservandone lo pseudocodice è possibile comprendere il ragionamento di fondo per ciascuna elaborazione effettuata.

3.1.1 Pseudocodice

Algorithm 1 Reducer

```
1: Initialize historical stock prices dictionary
2: for record in stdin do
3:     Set ticker, open_value, close_value, low, high, date
4:     Calcolo della data prima quotazione
5:     if date < historical_stocks_prices.ticker.first_quotation_date then
6:         Aggiorno i campi riguardanti la prima data
7:     end if
8:     Calcolo della data ultima quotazione
9:     if stock_date > historical_stocks_prices.ticker.last_quotation_date then
10:        aggiorno i campi riguardanti l'ultima data
11:    end if
12:    variation = calcolo della variazione percentuale
13:    Calcolo prezzo massimo e minimo
14:    if low < historical_stocks_price.ticker.min_price then
15:        aggiorno il minimo prezzo
16:    end if
17:    if high > historical_stocks_prices.ticker.max_price then
18:        aggiorno il massimo prezzo
19:    end if
20:    Giorni consecutivi di crescita del valore di chiusura
21:    if close_value > open_value then
22:        incremento il contatore temp_count > consecutive_days
23:        aggiorno i dati
24:    end if
25:    temp = 0 temp_count > consecutive_days
26:    aggiorno i campi
27:
28:    temp = 0
29:    ordino il dizionario rispetto all'ultima data
30: end for
```

3.1.2 Output

1. A 1999-11-18 00:00:00 2018-08-24 00:00:00 6498.0003 7.51073 115.879829 4 2005
2. AA 1970-01-02 00:00:00 2018-08-24 00:00:00 4243.9999 3.6045 117.194313 8 1980
3. AABA 1996-04-12 00:00:00 2018-08-24 00:00:00 6790.0002 0.645833 125.03125 11 2003
4. AAC 2018-01-16 00:00:00 2018-08-24 00:00:00 850.0 7.79 12.96 5 2018
5. AAL 2005-09-27 00:00:00 2018-08-24 00:00:00 3782.0 1.45 63.27 7 2018
6. AAME 1980-03-17 00:00:00 2018-08-24 00:00:00 170.00000000000003 0.375 15.8 4 2011
7. AAN 1987-01-20 00:00:00 2018-08-24 00:00:00 4949.0001999999995 0.481481 51.529999 9 2015
8. AAOI 2013-09-26 00:00:00 2018-08-24 00:00:00 4186.9999 8.08 103.410004 3 2014
9. AAON 1992-12-16 00:00:00 2018-08-24 00:00:00 4034.9998 0.089771 43.299999 6 2006
10 AAP 2001-11-29 00:00:00 2018-08-24 00:00:00 16336.000100000001 12.33 201.240005 7 2004

3.2 Implementazione Hive

L'implementazione in Hive sfrutta delle operazioni di riempimento tabellare per memorizzare le informazioni necessarie per l'elaborazione dei dati. L'implementazione sfrutta una serie di tabelle di preparazione che ci permettono di memorizzare le informazioni necessarie per rispondere agli obiettivi del Job, nello specifico si hanno le seguenti tabelle: (1) *docs* viene utilizzata per contenere l'intero dataset, (2) *ticker_first_date* permette di memorizzare la data della prima azione corrispondente al ticker e (3) *ticker_last_date* la data corrispondente all'ultima azione per quel ticker, (4) *ticker_min_low* e (5) *ticker_max_high* memorizzano rispettivamente il prezzo minimo e il prezzo massimo di ciascuna azione, (6) *ticker_fd* e (7) *ticker_ld* permettono di tenere in memoria i valori necessari per calcolare la variazione percentuale, (7) *history_stock_prices* è la tabella adibita al calcolo della variazione e infine vi è (8) *results* per la memorizzazione dei risultati. Infine tramite un'operazione di *select* andare a stampare a schermo il suo contenuto.

3.2.1 Output

1. NZF 2001-10-03 2018-08-24 -2.7796167163269487 7.3 16.4
2. RMCF 1986-02-11 2018-08-24 252.2041584157927 0.54112554 24.47619
3. SPEX 1991-04-08 2018-08-24 -99.99230801950998 0.83 57000.0
4. AKRX 2002-08-02 2018-08-24 1975.714308461365 0.25 57.1
5. RMBS 1997-05-14 2018-08-24 57.61983571958936 3.08 127.0
6. MNTA 2004-06-22 2018-08-24 247.63124576547537 4.58 34.6
7. HF 2007-01-31 2018-08-24 158.82952532135263 0.98554534 51.74
8. ISIG 1993-03-18 2018-08-24 -45.86666742960612 0.25 11.48
9. DNI 1998-06-24 2018-08-24 -79.30000305175781 7.6 62.5
10. NYT 1973-05-03 2018-08-24 1991.743244967093 0.5729167 53.0

3.3 Implementazione Spark

L'implementazione in Spark si basa sul linguaggio di programmazione Python. Vengono effettuate delle operazioni di filtraggio e pre-elaborazione per ottenere l'RDD relativo ai

tickers contenuti nel dataset. A questo punto tramite la funzione *job* verranno effettuate le stesse operazioni contenute all'interno dello script MapReduce, la differenza principale risiede nella manipolazione dei dati tramite l'utilizzo di della funzione *mapValues* che ci permette di utilizzare la funzione *job* per compiere il mapping e il reducing dei nostri dati.

3.3.1 Pseudocodice

Algorithm 2 Spark Job 1

```
RDD_input = open(historical_stock_prices)
RDD_tickers = grop by ticker RDD_input
Job(x), elabora il dataset seguendo il ragionamento del MapReduce
records.sortby = ordinamento per ultima data
output = variation_max join volume_max
```

3.3.2 Output

1. ('HASI', ('2013-04-18 00:00:00', '2018-08-24 00:00:00', 85.43859734246017, 25.2800006866455, 9.14999961853027))
2. ('ARKR', ('1985-12-13 00:00:00', '2018-08-24 00:00:00', 140.0, 43.4799995422363, 1.125))
3. ('JNJ', ('1970-01-02 00:00:00', '2018-08-24 00:00:00', 11054.871544471138, 148.320007324219, 0.776041686534882))
4. ('ATRC', ('2005-08-05 00:00:00', '2018-08-24 00:00:00', 142.47158788806414, 34.2200012207031, 1.0))
5. ('CBMG', ('2007-06-04 00:00:00', '2018-08-24 00:00:00', -18.599998474121193, 125.0, 1.0))
6. ('CSWC', ('1980-03-17 00:00:00', '2018-08-24 00:00:00', 1749.2120916193167, 51.9500007629395, 0.9375))
7. ('DRD', ('1994-05-16 00:00:00', '2018-08-24 00:00:00', -98.26122439637476, 155.0, 1.10000002384186))
8. ('GOL', ('2004-06-24 00:00:00', '2018-08-24 00:00:00', -71.89901164372529, 82.5, 0.479999989271164))
9. ('MDCA', ('1995-04-17 00:00:00', '2018-08-24 00:00:00', -39.99999761581425, 28.6499996185303, 0.666666686534882))
10. ('AFG', ('1980-03-17 00:00:00', '2018-08-24 00:00:00', 1732.3835338335539, 121.690002441406, 4.66666650772095))

3.4 Tempi tabellari e Grafici

	MapReduce	Hive	Spark
2M records	7.24 Minuti	8.10 Minuti	15.22 Minuti
100K records	3.20 secondi	1.25 Minuti	52 secondi
50K records	2.26 secondi	47 Secondi	34 secondi

Table 1: Tempi di esecuzione in locale

	MapReduce	Hive	Spark
2M records	2.53 Minuti	3.15 Minuti	6.48 Minuti
100K records	36 Secondi	1.25 Minuti	2.50 Minuti
50K records	20 Secondi	52 secondi	1.27 Minuti

Table 2: Tempi di esecuzione su cluster AWS a 3 Nodi

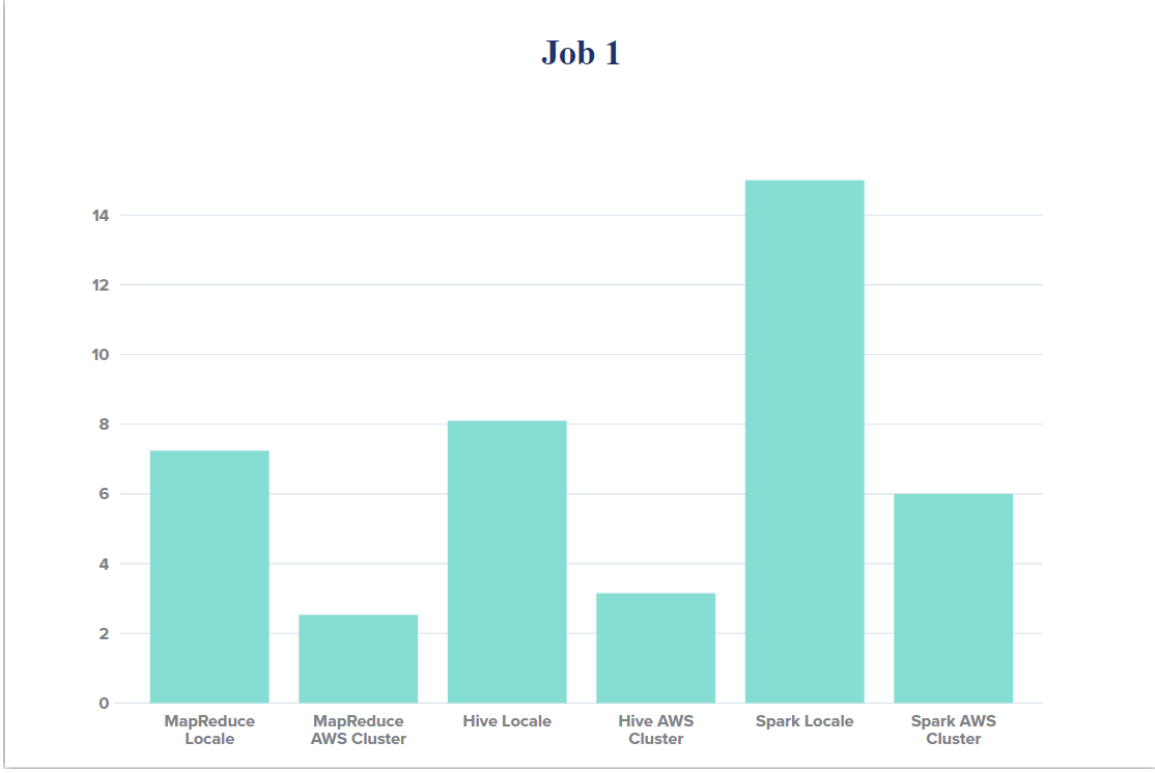

Figure 1: Confronto tempi di esecuzione in locale e su un cluster AWS a 3 nodi

4. Job 2

L'obiettivo di questo Job è quello di generare un report contenente, per ciascun settore e per ciascun anno del periodo 2009-2018: (1) la variazione percentuale della quotazione del settore nell'anno, (2) l'azione del settore che ha avuto il maggior incremento percentuale nell'anno (con indicazione dell'incremento) e (3) l'azione del settore che ha avuto il maggior volume di transazioni nell'anno (con indicazione del volume). Il report verrà ordinato per nome del settore.

4.1 Implementazione MapReduce

Questo job verte sull'analisi di entrambi i dataset a nostra disposizione. Per poterli sfruttare al meglio la soluzione implementata utilizza una coppia [mapper, reducer] sia per il primo dataset che per il secondo, A questo punto viene utilizzata una coppia [mapper, reducer] adibita all'operazione di Join tra i due output generati in modo tale da ottenere il punto di partenza per l'elaborazione d'interesse. La fase di elaborazione viene eseguita tramite una quarta coppia [mapper, reducer] che genererà l'output di interesse. Andando a riassumere dunque la soluzione implementata si ha che: (1) Si prepara il dataset *historical_stock_prices.csv* tramite un operazione di MapReduce, (2) Si prepara il dataset *historical_stocks.csv* tramite un operazione di MapReduce, (3) Si effettua il Join tra i due dataset usando un MapReduce apposito in modo tale da unire le informazioni di interesse, (4) L'elaborazione finale viene eseguita con un altro MapReduce che genererà l'output finale e complessivo.

4.1.1 Pseudocodice

Alcuni passaggi e componenti dell'implementazione del secondo Job sono stati omessi per ragioni di sintesi ed importanza. Tutto ciò che non viene mostrato come pseudocodice è consultabile in forma implementativa direttamente nel Github o nella codice sorgente che accompagna questa relazione.

Algorithm 3 Reducer Join

```
1:  Inizializza ticker_sector come dizionario
2:  ticker_data = []
3:  for record in stdin do
4:      if volume AND variation = 0 then
5:          Mi salvo il settore di appartenenza
6:      end if
7:      ELSE: Inserisco i dati in ticker_data
8:      for record in ticker_data do
9:          Aggiorno i campi
10:         if ticker in ticker_sector then
11:             aggiorno il campo sector ed invio in stdout
12:         end if
13:     end for
```

Algorithm 4 Reducer Elaborazione finale

```
1: Inizializza sector_year come dizionario
2: for record in input do
3:     if sector not in sector_year then
4:         Inizializza dizionario sector_year.sector
5:         Inizializza dizionario sector_year.sector.year
6:         Aggiorna i campi relativi
7:     end if
8:     if sector insector_year then:
9:         if yearnotinsector_year.sector then
10:            Inizializza dizionario sector_year.sector.year
11:            aggiorno i campi
12:        end if
13:    end if
14:    Else: incremento somme prime date e ultime
15:    if variation > sector_year.sector.year.variation then
16:        aggiorno
17:    end if
18:    if volume > sector_year.sector.year.volume then
19:        aggiorno
20:    end if
21: end for
22: Ordino il dataset
23: for Each result do
24:    for Each Year do
25:        Calcolo la variazione totale, il ticker corrispondente e il volume totale
```

4.1.2 Output

1. BASIC INDUSTRIES 2017 18.490788621316156 LAC 201.35593220338984 VALE 7023267600.0
2. BASIC INDUSTRIES 2010 17.646506027353933 BLD 519.8019801980198 VALE 6157781700.0
3. BASIC INDUSTRIES 2009 49.4240311716882 OMN 696.103896103896 VALE 7707779900.0
4. BASIC INDUSTRIES 2011 -17.92396365644517 NEU 57.23015952380953 VALE 4811206400.0
5. BASIC INDUSTRIES 2013 11.804060769435887 EVGN 280.07736943907156 VALE 4428233700.0
6. BASIC INDUSTRIES 2018 -3.876598827230245 VRS 70.60198301566433 VALE 3710091900.0
7. BASIC INDUSTRIES 2015 -15.999189852751607 BLD 202.46731544283892 VALE 7139827700.0
8. BASIC INDUSTRIES 2014 -3.5079563560247022 BLD 884.6 VALE 5660183200.0
9. BASIC INDUSTRIES 2012 9.820973647284616 LPX 134.46601941747574 VALE 4659766700.0
10. BASIC INDUSTRIES 2016 25.761078552219864 TECK 451.7906611570248 VALE 7602388100.0

4.2 Implementazione Hive

La prima table viene eseguito il join per associare il settore ad ogni ticker poi viene applicato un filtraggio degli anni. Le successive due table servono per il calcolo della prima e ultima data della quotazione per ogni anno, per ogni ticker. Le successive due table servono per il calcolo del volume totale del settore e della variazione percentuale annua di ogni ticker. Successivamente viene effettuato un join per calcolare la vaariazione percentuale annua di ogni settore di ogni anno. Nelle ultime due tabelle vengono effettuati i join per l'output
Dato che il dataset historical_stocks contiene delle virgole all'interno dei propri campi è stato deciso di compiere una pre-elaborazione di questo file attraverso un MapReduce dedicato, il quale andrà a ripulire il nostro file permettendoci di utilizzarlo correttamente all'interno di Hive, il quale altrimenti non sarebbe stato in grado di recuperare il nome per intero. Questa pre-elaborazione è stata eseguita anche in occasione del Job 3 durante l'implementazione in Hive.

4.2.1 Output

1. BASIC INDUSTRIES OMN 696.1039306864535 7707779900 49.424031175368874 2009
2. BASIC INDUSTRIES BLD 519.8019851061565 6157781700 17.646505894092083 2010
3. BASIC INDUSTRIES NEU 57.230159214564736 4811206400 -17.923963733424003 2011
4. BASIC INDUSTRIES LPX 134.46602222663054 4659766700 9.82097336586815 2012
5. BASIC INDUSTRIES EVGN 280.0773564517266 4428233700 11.80406061456878 2013
6. BASIC INDUSTRIES BLD 884.5999717712402 5660183200 -3.5079563269673564 2014
7. BASIC INDUSTRIES BLD 202.46730996167304 7139827700 -15.999189842380105 2015
8. BASIC INDUSTRIES TECK 451.7906614007032 7602388100 25.76107859079986 2016
9. BASIC INDUSTRIES LAC 201.35594705232606 7023267600 18.49078865005883 2017
10. BASIC INDUSTRIES VRS 70.60198417759054 3710091900 -3.876598851730691 2018

4.3 Implementazione Spark

Dopo aver caricato i due dataset viene eseguito un raggruppamento per ticker e per anno, viene utilizzata una map per il calcolo della variation percentuale e del volume totale per ticker e anno. Viene effettuato un join per associare il settore al ticker. Vengono poi utilizzate due map per il calcolo della variazione massima per ogni settore e rispettivo volume totale. Sono impiegate altre due map per il calcolo la somma dei valori di chiusura per la prima data e ultima per sector e year. L'ultima map serve per calcolare la variazione percentuale annuale per ogni settore e viene poi stampato l'output.

4.3.1 Pseudocodice

Algorithm 5 Spark Job2

1: hsp = open(historical_stock_prices)
2: hs = open(historical_stocks)
3: records.groupby = hsp group by ticker and year
4: sectors = hs group by ticker
5: records.map = calcolo della variation e del volume totale per ticker e year
6: join = records join sectors on ticker
7: variation_max.map = calcolo delle variation_max per sector e year a partire dal join
8: volume_max.map = calcolo dei volumi totali per sector e year a partire join
9: first_date_close.map = calcolo la somma dei valori di chiusura per la prima data per sector e year
10: last_date_close.map = calcolo la somma dei valori di chiusura per l'ultima data per sector e year
11: year_variation.map = calcolo della variation annuale del sector
12: output = variation_max join volume_max join year_variation sort by sector e year =0

4.3.2 Output

1. ('BASIC INDUSTRIES', 2009, 49.424031175368874, ('OMN', 696.1039306864535), ('VALE', 7707779900)
2. ('BASIC INDUSTRIES', 2010, 17.64650589409203, ('BLD', 519.8020087119552), ('VALE', 6157781700)
3. ('BASIC INDUSTRIES', 2011, 49.424031175368874, ('NEU', 57.23015952380953), ('VALE', 4428233700.0)
4. ('BASIC INDUSTRIES', 2012, 9.820973647284616, ('BLD', 884.6), ('VALE', 5660183200.0)
5. ('BASIC INDUSTRIES', 2013, 11.804060769435887, ('EVGN', 280.07736943907156), ('VALE', 4428233700.0)
6. ('BASIC INDUSTRIES', 2014, -3.5079563560247022, ('BLD', 884.6), ('VALE', 5660183200.0)
7. ('BASIC INDUSTRIES', 2015, -15.999189852751607, ('BLD', 202.46731544283892), ('VALE', 7139827700.0)
8. ('BASIC INDUSTRIES', 2016, 25.761078552219864, ('TECK', 451.7906611570248), ('VALE', 7602388100.0)
9. ('BASIC INDUSTRIES', 2017, 25.00074140043379, ('OLED', 201.0462075343054), ('F',

9293424500.0)
10. ('BASIC INDUSTRIES', 2018, -3.876598827230245, ('VRS', 70.60198301566433), ('VALE', 3710091900.0)

4.4 Tempi Tabellari e Grafici

	MapReduce	Hive	Spark
2M records	7.35 Minuti	6.28 Minuti	6.44 Minuti
100K records	8.24 secondi	1.29 Minuti	36 secondi
50K records	5.13 secondi	43 secondi	24 secondi

Table 3: Tempi di esecuzione in locale

	MapReduce	Hive	Spark
2M records	5.24 Minuti	1.54 Minuti	1.28 Minuti
100K records	4.51 secondi	3.49 secondi	21 secondi
50K records	2.10 secondi.	1.8 secondi	17 secondi

Table 4: Tempi di esecuzione su cluster AWS a 3 Nodi

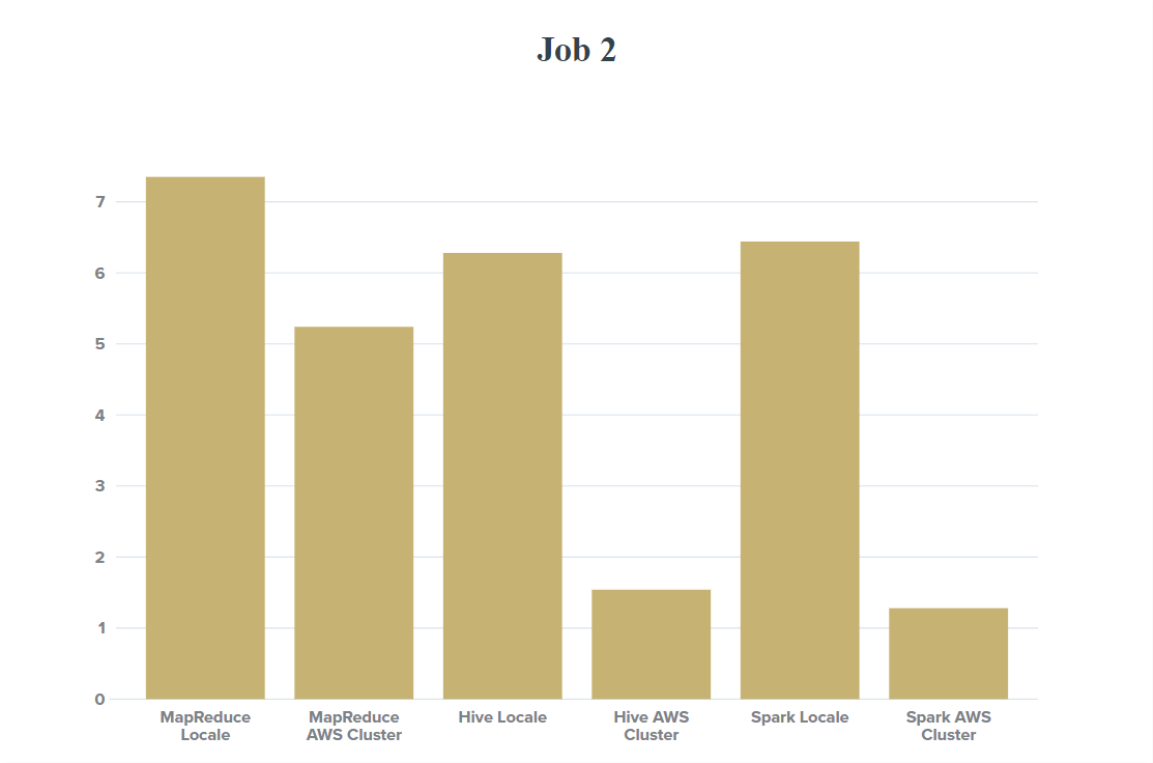

Figure 2: Confronto tempi di esecuzione in locale e su un cluster AWS a 3 nodi

5. Job 3

L'obiettivo di questo Job è quello di generare le coppie di aziende che si somigliano (sulla base di una soglia scelta in modo arbitrario) in termini di variazione percentuale mensile nell'anno 2017 mostrando l'andamento mensile delle due aziende.

5.1 Implementazione MapReduce

Questo Job è caratterizzato da una struttura di fondo molto simile al Job precedente, si è deciso di sfruttare questa somiglianza andando a riproporre la stessa soluzione implementata in precedenza. Questa soluzione prevede quattro step di MapReduce, nello specifico: (1) Si prepara il dataset *historical_stock_prices.csv* (2) Si prepara il dataset *historical_stocks.csv* (3) Si effettua il Join tra i due dataset, (4) Si esegue il MapReduce finale.

5.1.1 Pseudocodice

Alcuni passaggi e componenti dell'implementazione del secondo Job sono stati omessi per ragioni di sintesi ed importanza. Tutto ciò che non viene mostrato come pseudocodice è consultabile in forma implementativa direttamente nel codice sorgente che accompagna questa relazione. Il mapper di join è pressoché identico a quello implementato nella soluzione del Job 2. E' necessario porre l'attenzione sull'ultimo reducer, dato che la sua funzione *check* verrà ripresa anche nell'implementazione basata su Spark.

Algorithm 6 Reducer Finale

```
1: Inizializzo la lista dei mesi, soglia e dizionario name1_month
2: Function check(name1, name2)
3: temp = []
4: for month_a in name1 do
5:     for month_b in name2 do
6:         if month_a == month_b then
7:             Calcolo modulo differenza
8:             if 0 <= modulo_diff <= soglia then
9:                 pair_names = [name1, name2]
10:                if pair_names not in temp then
11:                    temp.append(pair_names)
12:                end if
13:            end if
14:        end if
15:    end for
16: end for
17: for record in input do
18:    if name not in name_month then
19:        Inizializzo il dizionario name_month.name
20:        name_month.name.month = [name,variation]
21:    end if
22:    if month not in name_month.name then
23:        name_month.name.month = [name,variation]
24:    end if
25: end for
26: for i AND j in name_month do
27:    check(i,j)
28: end for
```

5.1.2 Output

1. {180 DEGREE CAPITAL CORP.,1ST CONSTITUTION BANCORP (NJ)}
2. LUG:180 DEGREE CAPITAL CORP. -1.8404890477013123
3. LUG:1ST CONSTITUTION BANCORP (NJ) -0.8670498563310302
4. GIU:180 DEGREE CAPITAL CORP. 3.184710232218655
5. GIU:1ST CONSTITUTION BANCORP (NJ) 2.3188383682915847
6. {180 DEGREE CAPITAL CORP.,1ST SOURCE CORPORATION}
7. GIU:180 DEGREE CAPITAL CORP. 3.184710232218655
8. GIU:1ST SOURCE CORPORATION 2.4358961714378538
9. {180 DEGREE CAPITAL CORP.,3M COMPANY}
10. DIC:180 DEGREE CAPITAL CORP. -2.4752451875443477

5.2 Implementazione Hive

Come per gli altri job nella prima tabella viene effettuato un filtraggio sulla base del valore date. In questo caso viene preso in considerazione l'anno 2017. Vengono poi utilizzate due tabelle per il calcolo della prima e ultima data della quotazione per ogni mese, per ogni ticker. Successivamente viene svolto prima un join per associare il nome dell'azienda al ticker, poi viene calcolata la variazione percentuale mensile. Infine l'ultima tabella per il calcolo delle coppie con variazione percentuale mensile simile, sulla base di un valore soglia, attraverso un join della tabella precedente con se stessa con relativi vincoli necessari.

5.2.1 Output

1. 180 DEGREE CAPITAL CORP. 1ST CONSTITUTION BANCORP (NJ) Jul TURN -1.8404890477013416 Jul FCCY -0.8670498563314385
2. 180 DEGREE CAPITAL CORP. 1ST CONSTITUTION BANCORP (NJ) Jun TURN 3.1847102322189293 Jun FCCY 2.3188383682914404
3. 180 DEGREE CAPITAL CORP. 1ST SOURCE CORPORATION Jun TURN 3.1847102322189293 Jun SRCE 2.4358961714377148
4. 180 DEGREE CAPITAL CORP. 3M COMPANY Dec TURN -2.4752451875445414 Dec MMM -2.3968479890478402
5. 180 DEGREE CAPITAL CORP. 8X8 INC Dec TURN -2.4752451875445414 Dec EGHT -1.6050212383870353
6. 180 DEGREE CAPITAL CORP. A.H. BELO CORPORATION Dec TURN -2.4752451875445414 Dec AHC -3.0302954406081475
7. 180 DEGREE CAPITAL CORP. ABB LTD Sep TURN 6.097562482568367 Sep ABB 6.4058459005002
8. 180 DEGREE CAPITAL CORP. ABBOTT LABORATORIES Aug TURN 3.124996973201678 Aug ABT 3.6419065010355953
9. 180 DEGREE CAPITAL CORP. ABERDEEN GLOBAL PREMIER PROPERTIES FUNDJun TURN 3.1847102322189293 Jun AWP 2.589002082420456
10 180 DEGREE CAPITAL CORP. ABRAXAS PETROLEUM CORPORATION Oct TURN 9.248552881929333 Oct AXAS 9.230772428493685

5.3 Implementazione Spark

Vengono caricati i dataset in input, i record vengono raggruppati sulla base del valore ticker viene utilizzata una map per il calcolo della variazione percentuale mensile di ogni ticker per ciascun mese. Attraverso un join viene associato il nome dell'azienda a ciascun ticker, successivamente viene chiamata la funzione check, come quella implementata in MapReduce, per la verifica della somiglianza dei valori di variazione percentuale mensile tra due aziende.

5.3.1 Pseudocodice

Algorithm 7 Spark Job3

```
Hsp = open(historical_stock_prices)
Hs = open(historical_stocks)
Records.groupby = hsp group by ticker
Names = hsp group by ticker and name
Records.map = calcolo della variation per ticker e month
Join = records join name
for element_1 AND element_2 in join do
    Chiama la funzione check sulle variation
end for
```

5.3.2 Output

1. {"AMTRUST FINANCIAL SERVICES,CITIZENS HOLDING COMPANY}
2. GIU:AFSI 7.834760573066421
3. GIU:CIZN 8.108111537627796
4. {"AMTRUST FINANCIAL SERVICES,GREAT ELM CAPITAL CORP.}
5. GEN:AFSI -3.2269932282482516
6. GEN:GECC -3.6300785425412303
7. {"AMTRUST FINANCIAL SERVICES,"AVEO PHARMACEUTICALS}
8. SET:AFSI 8.199360869530434
9. SET:AVEO 7.98816484570356
10. {"AMTRUST FINANCIAL SERVICES,"DERMIRA}

5.4 Tempi Tabellari e Grafici

	MapReduce	Hive	Spark
2M records	5.36 Minuti	7.03 Minuti	9.17 Minuti
100K records	19 secondi	1.29 Minuti	33 Secondi
50K records	8 secondi	29 secondi	21 Secondi

Table 5: Tempi di esecuzione in locale

	MapReduce	Hive	Spark
2M records	2.53 Minuti	3.15 Minuti	7.14 Minuti
100K records	12 Secondi	1.02 Minuti	27 Secondi
50K records	10 Secondi	47 secondi	18 Secondi

Table 6: Tempi di esecuzione su cluster AWS a 3 Nodi

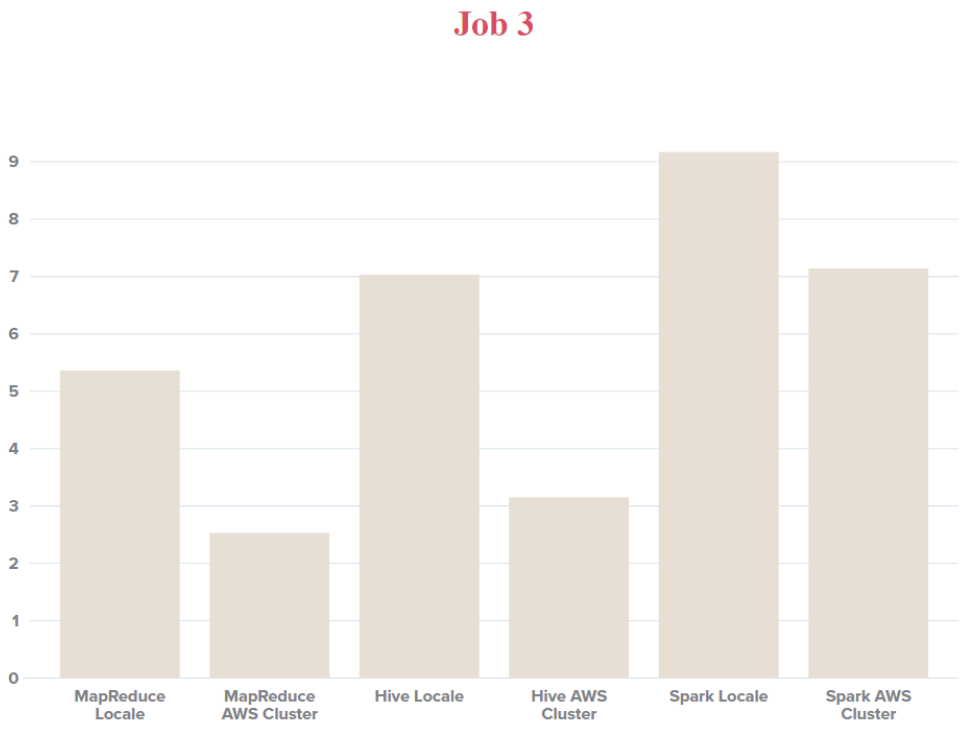

Figure 3: Confronto tempi di esecuzione in locale e su un cluster AWS a 3 nodi