**Rapporto finale del primo progetto Big Data**

Andrea Giaccone - Marco De Giovanni

440846 499633

**1- Specifiche scelte per lo svolgimento del progetto**

Le specifiche del progetto non definite nel documento consegnato e scelte dai sottoscritti sono le seguenti:

* Le singole compagnie vengono riconosciute tramite ticker, il ticker è stato preferito al nome per le seguenti ragioni:
  + Ufficialmente il ticker è univoco per ogni azienda (o meglio, è univoco per ogni azione, che è sempre legata ad una sola azienda), la presenza di più ticker per ogni azienda sarebbe dovuta al fatto che compagnie particolarmente grandi preferiscono “scorporarsi” in più aziende, con nomi molto simili, per gestire mansioni diverse (es **GOOG: *Alphabet Inc. Class C Capita*l - GOOGL: *Alphabet Inc. Class A Common Stock*** )
  + I nomi apparentemente possono contenere dei typo e/o degli errori di parsing, ciò non ci ha dato fiducia sul fatto che i nomi siano effettivamente univoci
* La variazione percentuale della quotazione, dato il prezzo di chiusura precedente (*min*) e antecedente (*max*) in base al range di tempo scelto, è il seguente: ***var =* ((*max-min)/min)\*100***
* I risultati generati dal Map Reduce e da Spark sono tabulati tramite il carattere TAB “\t”
* Sempre per i risultati generati dal Map Reduce e da Spark, i risultati delle operazioni numeriche con virgola non sono arrotondati dopo la virgola
* Nel secondo job i ticker etichettati con settore “N/A” vengono comunque presi in considerazione, trattando quindi N/A come fosse un settore a sè
* Nel terzo job la soglia è dell’1% e deve essere rispettata per tutti i mesi dell’anno, le coppie sono tra ticker

**2- Operazioni di pre-processing sui dati**

Sono state effettuate le seguenti operazioni di pulizia dei file CSV:

1. Per i job eseguiti con Mapreduce ed Hive sono state eliminate le prime righe dei due file CSV (in quanto erano i titoli delle colonne)
2. Nel file “historical\_stock\_price.csv” è stato notato che alcuni caratteri erano scritti con il formato di encoding HTML (Ad es. il carattere **‘** era scritto come &#39;), è stato quindi effettuato uno unescape su tutte le righe per inserire i caratteri nel formato corretto
3. Nel file “historical\_stock.csv” la tabulazione tramite virgole è stata cambiata con una tabulazione tramite TAB e sono state rimosse le virgolette sui nomi contenenti virgole, ciò è stato fatto per evitare errori al momento dello split dei dati
4. Nel file “historical\_stock\_price.csv” la tabulazione tramite virgole è stata cambiata con una tabulazione tramite TAB, ciò è stato fatto per uniformarsi alla tabulazione del file historical\_stock.csv

Le operazioni 2, 3 e 4 sono state effettuate attraverso uno script in python (csv\_cleaner.csv) incluso nei file consegnati e scritto per l’occasione.

**3- Svolgimento del primo job**

**3.1- MapReduce**

Viene eseguita una singola operazione di map reduce, dove nel mapper viene effettuato uno strip e una “select” dei dati ricevuti, mentre nel reducer vengono prima salvati in memoria i ticker associati ai valori richiesti (calcolando anche la variazione percentuale), dopodiché analizzati tutti i dati questi vengono stampati.

***mapper:***

*per ogni linea “line” in input da “historical\_stock\_price”:*

*line.strip()*

*ticker, open, close, abj\_close, lowThe, highThe, volume, date = line.split(“ \t ”)*

*stampa( ticker, close, low, high, date )*

**reducer:**

*tickers = []*

*per ogni ticker, close, low, high, date da “mapper”:*

*esegui:*

*close = parsing\_float(close)*

*low = parsing\_float(low)*

*high = parsing\_float(high)*

*date = parsing\_date(date)*

*errore parsing fallito:*

*scarta elemento*

*se ticker non è presente in tickers:*

*tickers.aggiungi(ticker)*

*tickers[ticker][minDate] = date*

*tickers[ticker][maxDate] = date*

*tickers[ticker][minDate][close] = close*

*tickers[ticker][maxDate][close] = close*

*tickers[ticker][varClose] = 0.0*

*altrimenti:*

*se tickers[ticker][minDate] > date:*

*tickers[ticker][minDate] = date*

*tickers[ticker][minDate][close] = close*

*tickers[ticker][varClose] = aggiorna con nuovo minDate*

*altrimenti se tickers[ticker][maxDate] < date:*

*tickers[ticker][maxDate] = date*

*tickers[ticker][maxDate][close] = close*

*tickers[ticker][varClose] = aggiorna con nuovo maxDate*

*#tutti i ticker sono stati processati a questo punto*

*stampa (tickers)*

**3.2- Hive**

Dopo aver importati i valori da “historical\_stock\_price” in una tabella:

1. Viene creata una tabella di supporto, dove vengono inseriti i due prezzi minimi e massimi raggiunti, nonché la data più lontana e vicina, tutto raggruppato per ogni ticker
2. Vengono create due ulteriori tabelle di supporto, in una viene inserito il valore di chiusura della data più lontana, nell’altra viene inserito il valore di chiusura della data più recente. La tabella di supporto del punto precedente può essere a questo punto droppata
3. Viene generata la tabella con i risultati eseguendo un join delle due tabelle di supporto generate in precedenza, calcolando la variazione in percentuale della quotazione

**3.3- Spark core**

L’idea di base è quella di utilizzare, come struttura dati per il chiave/valore per Spark, una tupla che ha come chiave il ticker e come valore una tupla di 5 elementi (variazione, data minima, data massima, prezzo minimo, prezzo massimo).

Su ogni RDD utilizzato si effettua quindi un’operazione di mapping per portare gli elementi nella struttura dati descritta sopra (ponendo a **None** gli elementi trascurabili), per poi effettuare come step finale una union di tutte le RDD, con successivo reduce per ottenere il risultato finale.

*rdd = importa da “historical\_stock\_price.csv”*

*flatMap su rdd = split TAB e parsing, se parsing fallisce mappa in [], poi mapping “line” -> (ticker, (close, min, high, date) )”*

*low\_rdd = reduce su rdd: X,Y = X se X.min < Y.min, altrimenti Y*

*mapping su low\_rdd: “line -> (ticker, (None, None, None, line.min, None ) )*

*high\_rdd = reduce di su rdd, X,Y = X se X.max > Y.max, altrimenti Y*

*mapping su high\_rdd: “line -> (ticker, (None, None, None, None, line.high ) )*

*mindate\_rdd = reduce su rdd: X,Y = X se X.date < Y.date, altrimenti Y*

*maxdate\_rdd = reduce su rdd: X,Y = X se X.date > Y.date, altrimenti Y*

*varclose\_rdd = reduce su mindate\_rdd* ***U*** *maxdate\_rdd: X,Y = (ticker, (variazione da X a Y se X.date < Y.date, altrimenti variazione da Y a X), None, None, None, None)*

*mapping su mindate\_rdd: “line” -> (ticker, (None, line.date, None, None, None)*

*mapping su maxdate\_rdd: “line” -> (ticker, (None, None, line.date, None, None)*

*elimina rdd*

*result\_rdd = reduce su (low\_rdd* ***U*** *high\_rdd* ***U*** *mindate\_rdd* ***U*** *maxdate\_rdd* ***U*** *varclose\_rdd): accorpa le righe eliminando i None*

*mapping su result\_rdd: da struttura dati chiave/valore a tabella*

*stampa result\_rdd*

**4- Svolgimento del secondo job**

**4.1- MapReduce**

In questo job sfruttiamo l’operazione di sort tra il mapper ed il reducer per far arrivare prima i risultati di un csv al reducer (historical\_stocks.csv), in modo da mettere i dati necessari in memoria e utilizzarli insieme agli elementi del csv successivo (historical\_stock\_price.csv).

***mapper:***

*per ogni linea “line” in input dal join:*

*line = line.strip.split*

*se lunghezza di line è pari a 5:*

*ticker, sector = line*

*stampa(“..”, ticker, sector)*

*altrimenti se lunghezza è pari a 8:*

*ticker, close, volume, date = line.split*

*se date è successivo al 2009:*

*stampa( ticker, date, close, volume)*

***reducer:***

*sectors = []*

*ticker\_sector = []*

*per ogni “line” data da “mapper”:*

*line = line.split*

*se lunghezza di line pari a 3 e primo elemento pari a “..”:*

*ticker, sector = line*

*ticker\_sector[ticker] = sector*

*passa direttamente al prossimo ciclo*

*ticker, date, close, volume = line*

*sector = ticker\_sector[ticker] = sector*

*esegui:*

*close = parsing\_float(close)*

*volume = parsing\_long(volume)*

*date = parsing\_date(date)*

*errore parsing fallito:*

*scarta elemento*

*se sector non è presente in sectors:*

*sectors.aggiungi(sector)*

*se l’anno del date non è presente in sectors[sector]:*

*sector[sector].aggiungi(date.year)*

*se ticker non è presente in sectors[sector][date.year]:*

*sectors[sector][date.year].aggiungi(ticker)*

*sectors[sector][date.year][ticker][minDate] = date*

*sectors[sector][date.year][ticker][maxDate] = date*

*sectors[sector][date.year][ticker][minDate][close] = close*

*sectors[sector][date.year][ticker][maxDate][close] = close*

*sectors[sector][date.year][ticker][varClose] = 0.0*

*sectors[sector][date.year][ticker][volume] = volume*

*altrimenti:*

*somma volume in sectors[sector][date.year][ticker][volume]*

*se sectors[sector][date.year][ticker][minDate] > date:*

*sectors[sector][date.year][ticker][minDate] = date*

*sectors[sector][date.year][ticker][minDate][close] = close*

*aggiorna sectors[sector][date.year][ticker][varClose]*

*altrimenti sectors[sector][date.year][ticker][maxDate] < date:*

*sectors[sector][date.year][ticker][maxDate] = date*

*sectors[sector][date.year][ticker][maxDate][close] = close*

*aggiorna sectors[sector][date.year][ticker][varClose]*

*per ogni settore “sector” in sectors:*

*per ogni anno “year” in sectors[sector]:*

*minDate\_sector = 0*

*maxDate\_sector = 0*

*best\_close\_ticker = ticker: “N/A”, close: 0.0*

*best\_volume\_ticker = ticker: “N/A”, volume: 0.0*

*per ogni ticker in sectors[sector][year]:*

*somma sectors[sector][year][ticker][minDate][close] a minDate\_sector*

*somma sectors[sector][year][ticker][maxDate][close] a maxDate\_sector*

*se sectors[sector][year][ticker][volume] > best\_volume\_ticker:*

*aggiorna best\_volume\_ticker con questo ticker*

*se sectors[sector][year][ticker][varClose] > best\_close\_ticker:*

*aggiorna best\_close\_ticker con questo ticker*

*calcola var\_close\_sector*

*stampa(sector, year, var\_close\_sector, best\_close\_ticker, best\_close\_volume)*

**4.2- Hive**

Importiamo i dataset historical\_stock\_prices e historical\_stocks in tabelle, poi:

1. Per ticker e per anno di interesse, calcoliamo la prima e l’ultima data e sommiamo i volumi
2. Estraiamo i close\_price di queste due date, per anno, per ticker
3. Utilizziamo i dati estratti per calcolare la variazione percentuale di quotazione per anno, per azione (***var =* ((*max-min)/min)\*100***)
4. Per settore, per anno, calcoliamo la variazione percentuale di quotazione con la formula sovrascritta, dove min e max sono ricavati come le somme dei close\_value delle prime ed ultime date delle aziende nell’anno di riferimento; e calcoliamo la massima variazione percentuale annua tra le aziende del settore e il volume massimo tra le aziende del settore, anno per anno
5. Estraiamo i ticker associati ai valori ricavati precedentemente del volume massimo e massima percentuale di quotazione per settore e per anno.

**4.3- Spark core**

Dopo aver effettuato un join di due RDD contenenti rispettivamente gli elementi di historical\_stock e historical\_stock\_price, vengono effettuati dapprima dei reduce su chiavi pari a tupe (ticker, sector, year), poi su chiavi pari a tuple (sector, year)

*price\_rdd = importa da historical\_stock\_price*

*flatMap su price\_rdd = split su TAB, poi parsing degli elementi, se parsing fallisce*

*oppure la data non è compresa tra 2009 e 2018 mappa come []*

*sector\_rdd = importa da historical\_stock*

*map su sector\_rdd = spli TAB, poi tupla (ticker, sector)*

*price\_rdd = price\_rdd join sector\_rdd*

*volume\_rdd = mappa price\_rdd come ((ticker, sector, year), volume), poi reduce by key con x,y: x+y, poi mappa come ((sector, year), (ticker, volume)), poi reduce by key con x,y: x se x.volume > y.volume, altrimenti y*

*xdate\_rdd = mappa price\_rdd come ((ticker, sector, year), (close, date))*

*elimina price\_rdd*

mindate\_rdd = reduce by key su xdate con x,y: x se x.date < y.date

maxdate\_rdd = reduce by key su xdate con x,y: x se x.date > y.date

elimina xdate\_rdd

*varclose\_rdd = mindate\_rdd* ***U*** *mindate\_rdd, poi redube by key con x,y: variazione*

*tra x e y, poi map come ((sector, year), (ticker, varclose)), poi reduce by key con x,y: x se x.varclose > y.varclose*

*mindate\_sector\_rdd = reduce by key su mindate\_rdd, con x,y: x+y*

*elimina mindate\_rdd*

*maxdate\_sector\_rdd = reduce by key su maxdate\_rdd, con x,y: x+y*

*elimina maxdate\_rdd*

*varclose\_sector\_rdd = mindate\_sector\_rdd* ***U*** *maxdate\_sector\_rdd, poi reduce by key con x,y: variazione tra x e y*

*elimina mindate\_sector\_rdd e maxdate\_sector\_rdd*

*mappa varclose\_rdd come ((sector, year), None, ticker, varcloseticker, None, None)*

*mappa volume\_rdd come ((sector, year), None, None, None, ticker, volumeticker)*

*mappa varclose\_sector\_rdd come ((sector, year), varclose, None, None, None, None)*

*varclose\_rdd* ***U*** *volume\_rdd* ***U*** *varclose\_sector\_rdd, poi reduce by key accorpanto le righe eliminando i none, poi map usando il tabulatore TAB, poi stampa*

**5- Svolgimento per terzo job**

**5.1- Map reduce**

Come anticipato prima, vengono effettuate due operazioni di map reduce, una prima di join per historical\_stock\_price e historical\_stock e una seconda con l’implementazione vera e propria delle logiche per l’esecuzione del job.

***mapper:***

*per ogni linea “line” ricevuta dal mapper “join”:*

*ticker, close, date = line.split*

*se anno del date è 2017:*

*stampa(ticker, close, date)*

***reducer:***

*tickers = [*

*1 = [],*

*2 = [].*

*…,*

*12 = []*

*]*

*threshold = X%*

*per ogni ticker, name, close, date ricevuto da “mapper”:*

*esegui:*

*close = parsing\_float(close)*

*date = parsing\_date(date)*

*errore parsing fallito:*

*scarta elemento*

*se ticker non è presente in tickers[date.month]:*

*tickers[date.month].aggiungi(ticker)*

*tickers[date.month][ticker][minDate] = date*

*tickers[date.month][ticker][maxDate] = date*

*tickers[date.month][ticker][minDate][close] = close*

*tickers[date.month][ticker][maxDate][close] = close*

*tickers[date.month][ticker][varClose] = 0.0*

*tickers[date.month][ticker][volume] = volume*

*altrimenti:*

*se tickers[date.month][ticker][minDate] > date:*

*tickers[date.month][ticker][minDate] = date*

*tickers[date.month][ticker][minDate][close] = close*

*tickers[date.month][date.year][ticker][varClose]*

*altrimenti se tickers[ticker][maxDate] < date:*

*tickers[date.month][ticker][maxDate] = date*

*tickers[date.month][ticker][maxDate][close] = close*

*aggiorna sectors[sector][date.year][ticker][varClose]*

*#Fase 0: Rimuovo le informazioni ormai superflue dalla memoria*

*per ogni mese “month”:*

*per ogni ticker in tickers[month]:*

*cancella tickers[month][ticker][minDate]*

*cancella tickers[month][ticker][maxDate]*

*#Fase 1: Cerco tutte le coppie che rispettano la soglia per il mese di Gennaio*

*couples = []*

*ord\_tickers = ordina\_per\_varClose(tickers[1])*

*per ogni ticker in ord\_tickers - 1:*

*succ\_ticker = elemento in lista successivo a ticker*

*finchè succ\_ticker esiste:*

*se varClose di ticker - varClose succ\_ticker rispetta soglia:*

*couple.aggiungi([ticker, succ\_ticker])*

*succ\_ticker = elemento in lista successivo a succ\_ticker*

*altrimenti:*

*esci*

*#Fase2: Elimino tutte le coppie che nei mesi successivi non rispettano più la soglia*

*per ogni coppia “couple” in couples:*

*per ogni mese “month” da Febbraio a Dicembre:*

*esegui:*

*se elementi in “couple” non rispettano la soglia nel mese “month”:*

*rimuovi “couple” da couples*

*errore varClose di quel mese non esistono per almeno un elemento:*

*rimuovi “couple” da couples*

*stampa(couples)*

**5.2- Hive**

Caricata la tabella con i dati di historical\_stock\_prices:

1. Creiamo una tabella di supporto con i ticker, la data minore, la data maggiore e il mese del 2017
2. Due tabelle per ricavare le tuple associate ai valori calcolati precedentemente
3. Creiamo una tabella con ticker, mese di riferimento, le date ed i prezzi di chiusura relativi alla prima e l’ultima tupla del mese e la variazione percentuale mensile dei ticker
4. Creiamo una tabella con le coppie di ticker simili, il mese di riferimento, e le loro variazioni percentuali mensili, facendo un join tra due copie della tabella al punto 3 e verificando che la differenza tra le variazioni percentuali delle due aziende nel mese sia minore della soglia fissata e che il primo ticker della coppia sia precedente alfabeticamente al secondo (per evitare di accoppiare le aziende con se stesse o creare coppie speculari)
5. Ed infine filtriamo la tabella al punto 4 per mantenere solamente le coppie che rispettano la differenza delle variazioni percentuali mensili per tutti e 12 i mesi.

**5.3- Spark core**

Dopo aver calcolato la variazione di ogni mese per ogni ticker, questi vengono accorpati in un unico RDD con chiave ticker e valore un array con le 12 variazioni. Fatto questo viene effettuato un prodotto cartesiano e il risultato infine filtrato. La scelta della struttura dati della RDD descritta sopra è dovuta anche alla volontà di mantenere il numero di elementi su cui effettuare il prodotto cartesiano quanto più basso possibile.

*price\_rdd = importa da historical\_stock\_price, poi flatmap con split da TAB, parsing (se parsing fallisce o l’anno non è 2017, mappa come []), infine porta nella forma ((ticker, month), (close, date))*

*mindate\_rdd = reduce by key da price\_rdd, con x,y: x se x.date < y.date, altrimenti y*

*maxdate\_rdd = reduce by key da price\_rdd, con x,y: x se x.date > y.date, altrimenti y*

*elimina price\_rdd*

*varclose\_rdd = redube by key su mindate\_rdd* ***U*** *maxdate\_rdd, con x,y: variazione tra x e y*

*elimina mindate\_rdd e maxdate\_rdd*

*varclose\_rdd = map su varclose\_rdd come (ticker, Array), dove Array è un array di dimensione 12 con tutti None ad accezione dell’elemento in posizione month-1, dove è invece presente la variazione relativa al mese dell’elemento*

*varclose\_rdd = reduce by key su varclose\_rdd, come vengono accorpate gli elementi eliminando i None nell’Array, poi prodotto cartesiano con se stesso (risultato: ((ticker1, varmesi1) , (ticker2, varmesi2)), poi filtro con le seguenti caratteristiche: stringa ticker1 < stringa ticker 2, threshold tra varmesi1 e varmes2 rispettata tra tutti gli elementi, poi mappa come “(ticker1, ticker2) (gennaio1, gennaio2), …, (dicembre1, dicembre2), poi stampa*

**6- Risultati**

Job 1: Map Reduce

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ticker** | **data prima quot.** | **data ultima quot.** | **variazione totale** | **minimo assoluto** | **massimo assoluto** |
| A | 1999-11-18 | 2018-08-24 | 109.636 % | 7.511 | 115.880 |
| AA | 1970-01-02 | 2018-08-24 | 508.325 % | 3.604 | 117.194 |
| AABA | 1996-04-12 | 2018-08-24 | 4910.910 % | 0.645 | 125.031 |
| AAC | 2018-01-16 | 2018-08-24 | 4.856 % | 7.790 | 12.960 |
| AAL | 2005-09-27 | 2018-08-24 | 111.140 % | 1.450 | 63.270 |
| AAME | 1980-03-17 | 2018-08-24 | -29.870 % | 0.375 | 15.800 |
| AAN | 1987-01-20 | 2018-08-24 | 4683.263 % | 0.481 | 51.530 |
| AAOI | 2013-09-26 | 2018-08-24 | 330.421 % | 8.080 | 103.410 |
| AAON | 1992-12-16 | 2018-08-24 | 41348.203% | 0.089 | 43.300 |
| AAP | 2001-11-29 | 2018-08-24 | 1084.150 % | 12.330 | 201.240 |

Job 1: Hive (SELECT \* FROM RESULT LIMIT 10)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ticker** | **data prima quot.** | **data ultima quot.** | **variazione totale** | **minimo assoluto** | **massimo assoluto** |
| AHH | 2013-05-08 | 2018-08-24 | 37.0 % | 8.78 | 16.17 |
| APO | 2011-03-30 | 2018-08-24 | 91.0 % | 8.85 | 37.35 |
| PEZ | 2006-10-12 | 2018-08-24 | 133.0 % | 11.79 | 60.17 |
| CRCM | 2014-01-24 | 2018-08-24 | -19.0 % | 4.89 | 29.25 |
| GTN | 2002-08-30 | 2018-08-24 | 45.0 % | 0.16 | 18.07 |
| FLWS | 1999-08-03 | 2018-08-24 | -31.0 % | 0.85 | 23.1875 % |
| GHDX | 2005-09-29 | 2018-08-24 | 391.0 % | 8.56 | 59.49 |
| VIAV | 1993-11-17 | 2018-08-24 | 878.0 % | 0.959 | 698.16 |
| MHD | 1997-05-06 | 2018-08-24 | -1.0 % | 7.5 | 19.99 |
| AMSWA | 1983-02-28 | 2018-08-24 | 283.0 % | 1.05 | 24.4375 |

Job 1: Spark core (prime 10 righe del file part-00000)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ticker** | **data prima quot.** | **data ultima quot.** | **variazione totale** | **minimo assoluto** | **massimo assoluto** |
| GHDX | 2005-09-29 | 2018-08-24 | 390.894 % | 8.560 | 59.499 |
| SBGL | 2013-02-11 | 2018-08-24 | -62.300 % | 2.020 | 20.166 |
| ITEQ | 2015-11-04 | 2018-08-24 | 42.180 % | 19.790 | 36.099 |
| TIPT | 2010-10-15 | 2018-08-24 | 35.611 % | 4.000 | 9.970 |
| RZA | 2012-08-24 | 2018-08-24 | 5.085 % | 23.080 | 30.870 |
| BSRR | 1994-04-18 | 2018-08-24 | 1097.600 % | 2.500 | 34.700 |
| GER | 2014-09-26 | 2018-08-24 | -66.099 % | 3.210 | 20.070 |
| AVGO | 2009-08-06 | 2018-08-24 | 1191.718 % | 14.330 | 279.770 |
| AMSR | 2005-11-18 | 2018-08-24 | 612.222 % | 8.120 | 66.470 |
| RARX | 2016-10-26 | 2018-08-24 | -14.153 % | 4.780 | 25.445 |

Job 2: Map Reduce

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **settore** | **anno** | **var. del settore** | **ticker miglior variazione** | **miglior variazione** | **ticker miglior volume** | **miglior volume** |
| BASIC INDUSTRIES | 2018 | -3.079 % | XRM | 213,817 % | VALE | 3710091900 |
| BASIC INDUSTRIES | 2013 | 10.322 % | XRM | 416.928 % | VALE | 4428233700 |
| BASIC INDUSTRIES | 2012 | -68.788 % | PATK | 261.860 % | VALE | 4659766700 |
| BASIC INDUSTRIES | 2016 | 13.829 % | TECK | 451.790 % | FCX | 10464699500 |
| BASIC INDUSTRIES | 2009 | 3.483 % | GURE | 709.722 % | FCX | 9141685400 |
| BASIC INDUSTRIES | 2014 | -71.902 % | BLD | 884.900 % | VALE | 5660183200 |
| BASIC INDUSTRIES | 2017 | 15.279 % | OPNT | 310.178 % | VALE | 7023267600 |
| BASIC INDUSTRIES | 2015 | -48.101 % | SUM | 35191.630 % | FCX | 7286761300 |
| BASIC INDUSTRIES | 2011 | -58.600 % | ROAD | 188.704 % | FCX | 5150807800 |
| BASIC INDUSTRIES | 2010 | 21.790 % | BLD | 519.802 % | FCX | 6891808600 |

Job 2: Hive (SELECT \* FROM RESULT LIMIT 10)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **settore** | **anno** | **var. del settore** | **ticker miglior variazione** | **miglior variazione** | **ticker miglior volume** | **miglior volume** |
| BASIC INDUSTRIES | 2009 | 3.483 % | GURE | 709.722 % | FCX | 9141685400 |
| BASIC INDUSTRIES | 2010 | 21.790 % | BLD | 519.802 % | FCX | 6891808600 |
| BASIC INDUSTRIES | 2011 | -58.600 % | ROAD | 188.704 % | FCX | 5150807800 |
| BASIC INDUSTRIES | 2012 | -68.788 % | PATK | 261.860 % | VALE | 4659766700 |
| BASIC INDUSTRIES | 2013 | 10.322 % | XRM | 416.928 % | VALE | 4428233700 |
| BASIC INDUSTRIES | 2014 | -71.902 % | BLD | 884.900 % | VALE | 5660183200 |
| BASIC INDUSTRIES | 2015 | -48.101 % | SUM | 35191.630 % | FCX | 7286761300 |
| BASIC INDUSTRIES | 2016 | 13.829 % | TECK | 451.790 % | FCX | 10464699500 |
| BASIC INDUSTRIES | 2017 | 15.279 % | OPNT | 310.178 % | VALE | 7023267600 |
| BASIC INDUSTRIES | 2018 | -3.079 % | XRM | 213,817 % | VALE | 3710091900 |

Job 2 Spark core (prime 10 righe nei primi file part-XXXXX)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **settore** | **anno** | **var. del settore** | **ticker miglior variazione** | **miglior variazione** | **ticker miglior volume** | **miglior volume** |
| TECHNOLOGY | 2014 | 61.198 % | KE | 60000 % | AAPL | 15914488100 |
| FINANCE | 2012 | 0.208 % | SNFCA | 516.513 % | BAC | 49413078400 |
| CONSUMER DURABLES | 2013 | -11.347 % | SGLB | 1184.722% | GT | 1069123300 |
| MISCELLANEOUS | 2016 | 27.406 % | GRVY | 224.615 % | BABA | 3725974300 |
| PUBLIC UTILITIES | 2009 | 2.245 % | EMP | 916.209 % | S | 12014127300 |
| N/A | 2011 | -15.043 % | CSF | 61.911 % | FTAI | 271666111000 |
| CAPITAL GOODS | 2009 | 31.081 % | NFEC | 1900 % | F | 19263087300 |
| MISCELLANEOUS | 2014 | 26.599 % | OMNT | 283.333 % | EBAY | 7378587900 |
| N/A | 2016 | -3.480 % | OPP | 36780 % | QQQ | 7115729700 |
| CAPITAL GOODS | 2016 | 9.363 % | SID | 251.086 % | F | 8985684000 |

Job 3 Map reduce

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **tickers** | **GEN** | **FEB** | **MAR** | **APR** | **MAG** | **GIU** | **LUG** | **AGO** | **SET** | **OTT** | **NOV** | **DIC** |
| CFS  CSA | -1.326  -0.939 | 1.124  1.144 | -1.644  -1.411 | 3.457  3.467 | -2.892  -3.055 | 1.122  1.499 | -0.763  -0.746 | -3.112  -2.624 | 6.684  6..872 | 0.013  0.199 | 4.017  4.013 | -0.287  0.206 |
| CFS  FYX | -1.326  -0.572 | 1.124  0.965 | -1.644  -1.645 | 3.457  3.250 | -2.892  -3.213 | 1.122  1.738 | -0.763  0.214 | -3.112  -2.179 | 6.684  5.965 | 0.013  -0.435 | 4.017  3.777 | -0.287  0.592 |
| CSA  FYX | -0.939  -0.572 | 1.144  0.965 | -1.411  -1.645 | 3.467  3.250 | -3.055  -3.213 | 1.499  1.738 | -0.746  0.214 | -2.624  -2.179 | 6.872  5.965 | 0.199  -0.435 | 4.013  3.777 | -0.206  0.592 |
| GTYHU  SNLN | -0.925  -0.532 | 0.577  0.268 | -1.714  -0.801 | 0.775  -0.107 | 0.038  -0.107 | -0.755  -0.809 | 0.478  0.217 | -0.095  -0.758 | -0.381  -0.163 | 0.000  0.109 | -0.469  -0.599 | -0.827  -0.274 |
| HYDN  AGND | -0.791  -0.068 | -0.657  -0.500 | -1.174  -0.523 | 0.577  0.344 | -0.857  -0.414 | -0.480  -0.069 | -0.240  -0.023 | -1.780  -1.173 | 0.438  0.816 | 0.969  0.184 | 0.289  -0.618 | -0.388  0.115 |
| HYDN  HYZD | -0.791  0.083 | -0.657  0.334 | -1.174  -0.787 | 0.577  0.336 | -0.857  0.125 | -0.480  -0.414 | -0.240  0.249 | -1.780  -1.117 | 0.438  0.835 | 0.969  0.165 | 0.289  -0.618 | -0.388  0.332 |
| SCACU  FTSL | -0.765  -0.082 | 1.050  0.371 | -0.292  -0.575 | 0.850  0.020 | 0.046  0.227 | -0.176  -0.411 | 1.351  0.600 | -0.654  -0.514 | -0.262  -0.331 | 0.094  0.290 | -0.936  -0.475 | -1.081  -0.186 |
| TCRX  MCV | -0.660  -0.040 | 0.297  1.120 | -0.890  -0.118 | -0.583  -0.354 | -0.198  0.433 | -0.329  -0.549 | 0.164  0.317 | 0.078  0.553 | -1.794  -1.139 | 0.194  -0.160 | 0.671  0.356 | -0.981  -0.158 |
| BDNX  JMPB | -0.628  -0.039 | 0.988  0.752 | 0.092  -0.353 | 0.184  1.104 | 0.718  -0.117 | -0.403  -0.972 | 0.350  0.117 | 0.531  -0.312 | -0.346  -0.273 | 0.734  0.788 | 0.145  0.864 | -1.289  -1.633 |
| BDNX  VBNB | -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  0.539 | -0.346  -0.663 | 0.734  -0.130 | 0.145  -0.575 | -1.289  -1.137 |

Job 3 Hive (SELECT \* FROM RESULT LIMIT 10)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ticker 1** | **ticker 2** | **mese** | **variazione t. 1** | **variazione t. 2** |
| AAXJ | AIA | 1 | 6.346 % | 6.454 % |
| AAXJ | AIA | 2 | 2.589 % | 1.925 % |
| AAXJ | AIA | 3 | 2.843 % | 2.618 % |
| AAXJ | AIA | 4 | 1.277 % | 1.205 % |
| AAXJ | AIA | 5 | 3.632 % | 3.929 % |
| AAXJ | AIA | 6 | 0.103 % | 0.811 % |
| AAXJ | AIA | 7 | 4.746 % | 5.208 % |
| AAXJ | AIA | 8 | 1.151 % | 0.543 % |
| AAXJ | AIA | 9 | -0.299 % | 0.392 % |
| AAXJ | AIA | 10 | 3.904 % | 4.898 % |

Job 3 Spark core (prime 10 righe nei primi file part-XXXXX)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **tickers** | **GEN** | **FEB** | **MAR** | **APR** | **MAG** | **GIU** | **LUG** | **AGO** | **SET** | **OTT** | **NOV** | **DIC** |
| EMCB  VGIT | 0.659  0.281 | 1.332  0.499 | -0.181  0.626 | 0.671  0.982 | 0.056  0.821 | -0.596  -0.293 | 0.657  0.590 | 1.038  0.755 | 0.192  -0.676 | -0.602  -0.123 | -0.273  -0.295 | 0.467  -0.311 |
| CFA  FEX | 1.426  1.630 | 3.993  3.953 | -1.340  -1.676 | 1.577  1.381 | 1.088  1.150 | 0.250  -0.337 | 0.882  1.483 | -0.467  -0.831 | 2.240  1.893 | 1.655  1.560 | 4.333  3,673 | 0.702  0.948 |
| CFA  CFO | 1.426  1.404 | 3.993  3.870 | -1.340  -1.275 | 1.577  1.693 | 1.088  1.112 | 0.250  0.022 | 0.882  1.071 | -0.467  -0.413 | 2.240  2.220 | 1.655  1.657 | 4.333  4.173 | 0.702  0.888 |
| EMCB  VMBS | 0.659  0.057 | 1.332  0.651 | -0.181  0.478 | 0.671  0.438 | 0.056  0.647 | -0.596  -0.246 | 0.657  0.628 | 1.038  0.607 | 0.192  -0.170 | -0.602  -0.075 | -0.273  -0.171 | 0.467  -0.114 |
| EMCB  SLQD | 0.659  0.338 | 1.332  0.358 | -0.181  0.159 | 0.671  0.317 | 0.056  0.317 | -0.596  0.039 | 0.657  0.575 | 1.038  0.177 | 0.192  0.000 | -0.602  -0.079 | -0.273  -0.198 | 0.467  -0.020 |
| EMCB  VCSH | 0.659  0.441 | 1.332  0.364 | -0.181  0.163 | 0.671  0.288 | 0.056  0.501 | -0.596  0.087 | 0.657  0.463 | 1.038  0.261 | 0.192  -0.012 | -0.602  -0.062 | -0.273  -0.250 | 0.467  -0.276 |
| EMCB  VWOB | 0.659  1.079 | 1.332  1.261 | -0.181  0.580 | 0.671  1.208 | 0.056  0.761 | -0.596  -0.521 | 0.657  0.916 | 1.038  1.682 | 0.192  0.061 | -0.602  0.160 | -0.273  -0.484 | 0.467  -0.087 |
| CFA  DWPP | 1.426  0.743 | 3.993  4.21 | -1.340  -1.315 | 1.577  0.622 | 1.088  1.264 | 0.250  -0.065 | 0.882  1.693 | -0.467  0.114 | 2.240  2.220 | 1.655  1.401 | 4.333  3,415 | 0.702  1.196 |
| CFA  KRMA | 1.426  1.105 | 3.993  4.093 | -1.340  -1.278 | 1.577  1.714 | 1.088  1.362 | 0.250  0.533 | 0.882  0.544 | -0.467  -0.827 | 2.240  2.823 | 1.655  1.274 | 4.333  3,392 | 0.702  1.114 |
| EMCB  MBB | 0.659  0.037 | 1.332  0.603 | -0.181  0.481 | 0.671  0.347 | 0.056  0.685 | -0.596  -0.326 | 0.657  0.743 | 1.038  0.579 | 0.192  -0.214 | -0.602  -0.075 | -0.273  -0.180 | 0.467  0.169 |

**7- Benchmark**

Nei grafici mostrati in seguito si prende come metro di misura il file “historical\_stock\_prices.csv” (input 1x, se raddoppiato 2x ecc.), se non specificato si intendono dati in input i csv dati in consegna preprocessati.

Il tempo di esecuzione è stato misurato prendendo il parametro “real” del comando time di Linux.

**7.1- Benchmark in locale**

Tutti i test in locale sono stati eseguiti su una VM, le caratteristiche della macchina host sono le seguenti:

**CPU:** AMD Ryzen 7 3700U, 4 cores / 8 threads @ 2.3Ghz

**RAM:** 8GB DDR4 @ 2400Mhz

**Storage:** SSD nvme

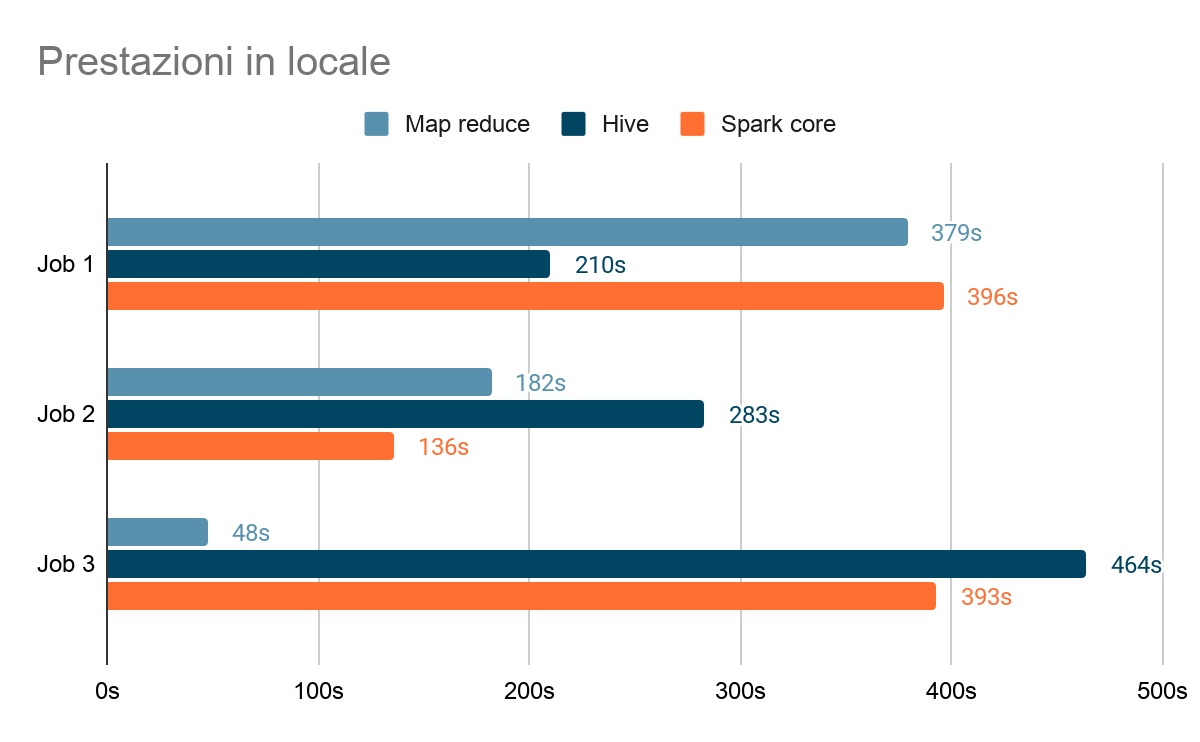
Per la VM le caratteristiche sono invece le seguenti:

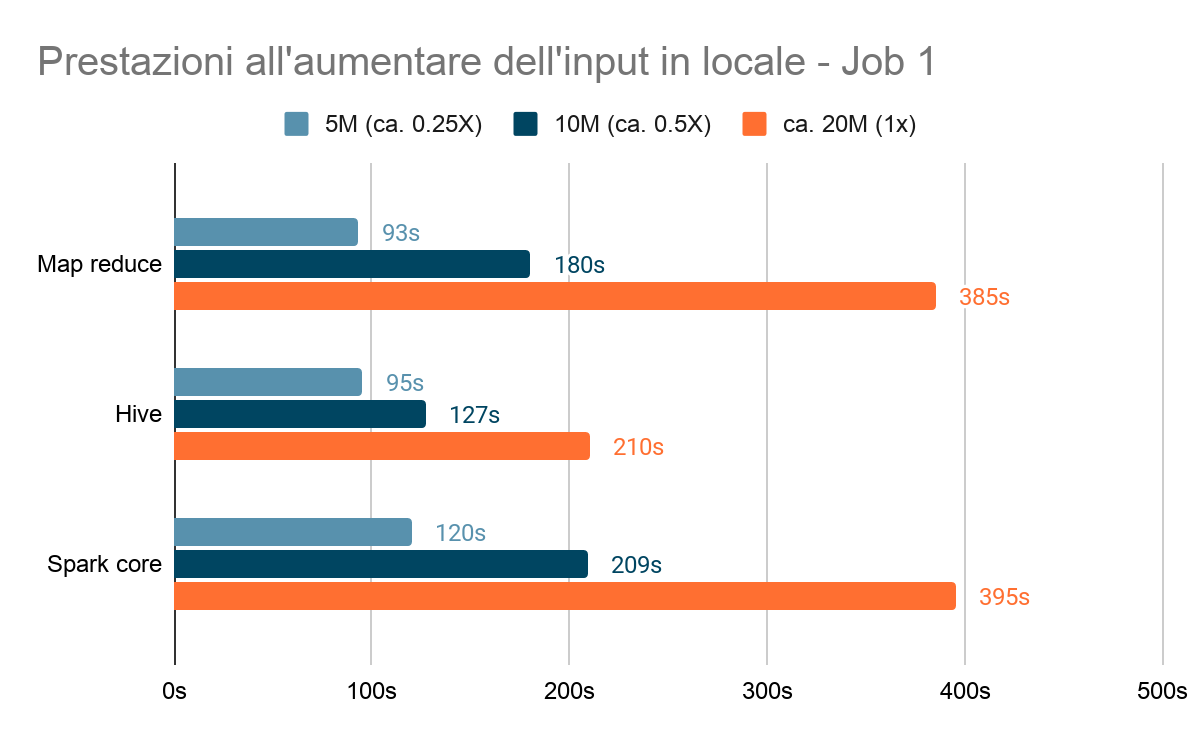
**Hypervisor:** Oracle VM VirtualBox

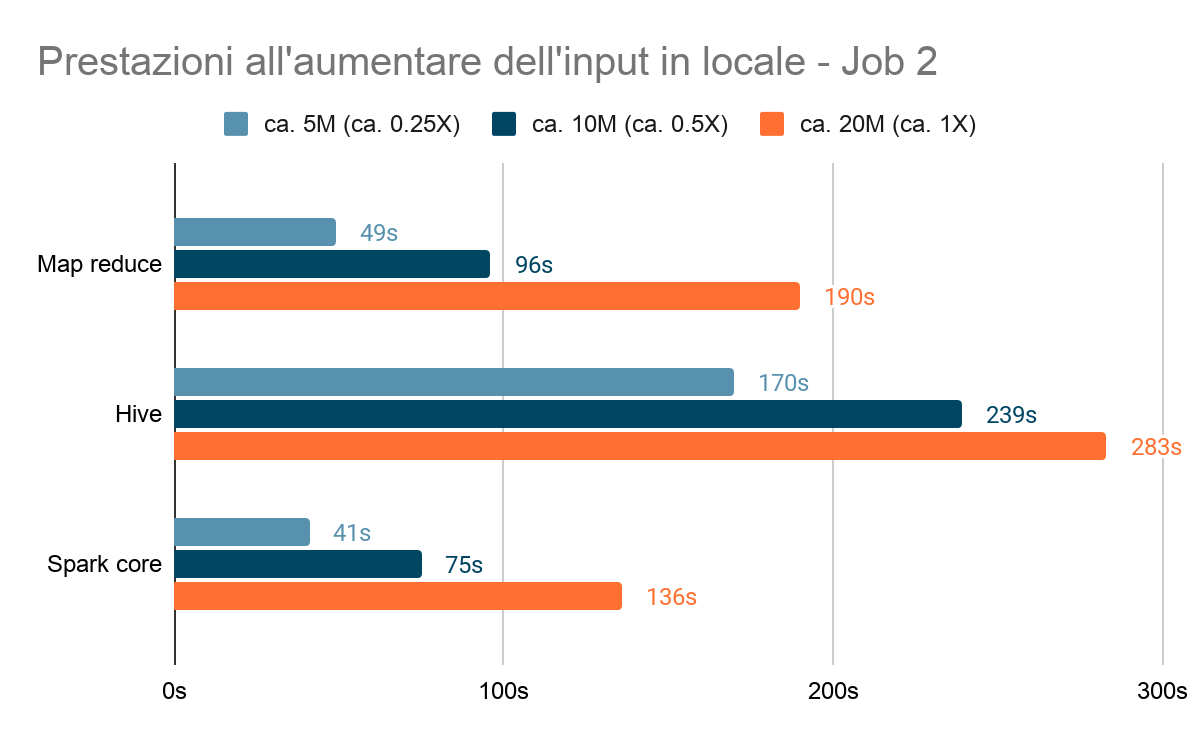
**OS:** Ubuntu 20.04.2 LTS ( con GNOME installato come GUI)

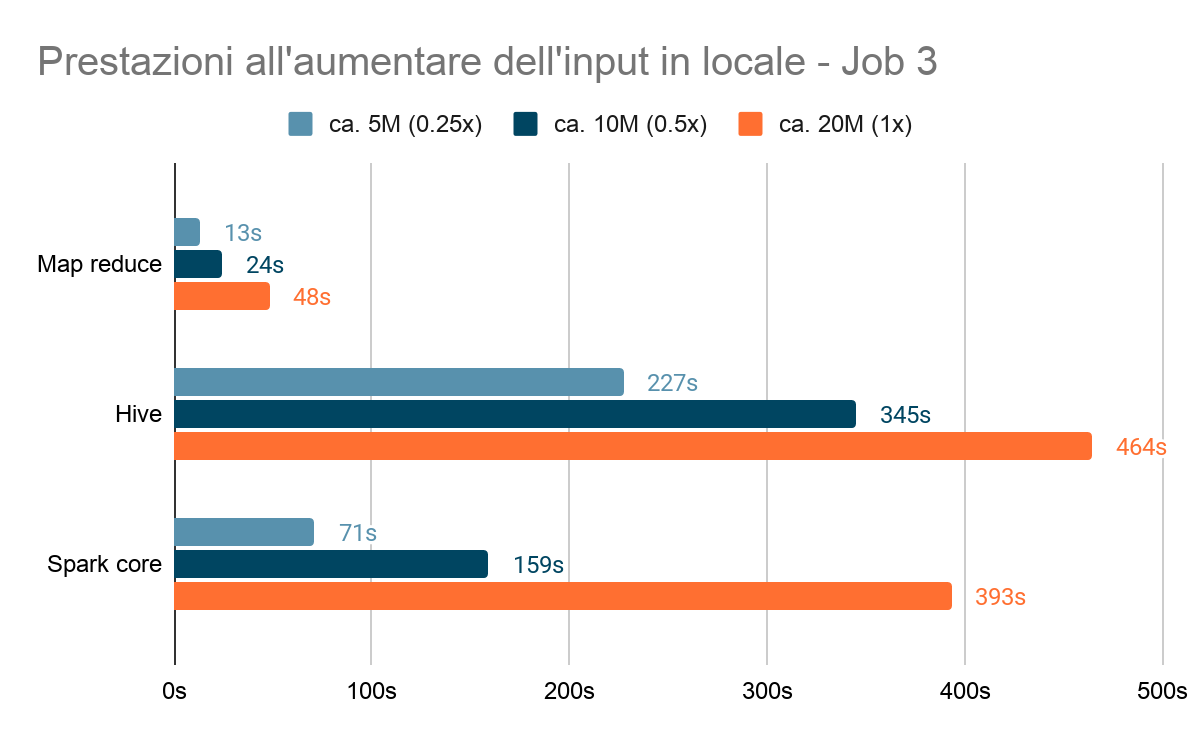
**vCPU:** 4 vCPU

**RAM:** 4GB





**

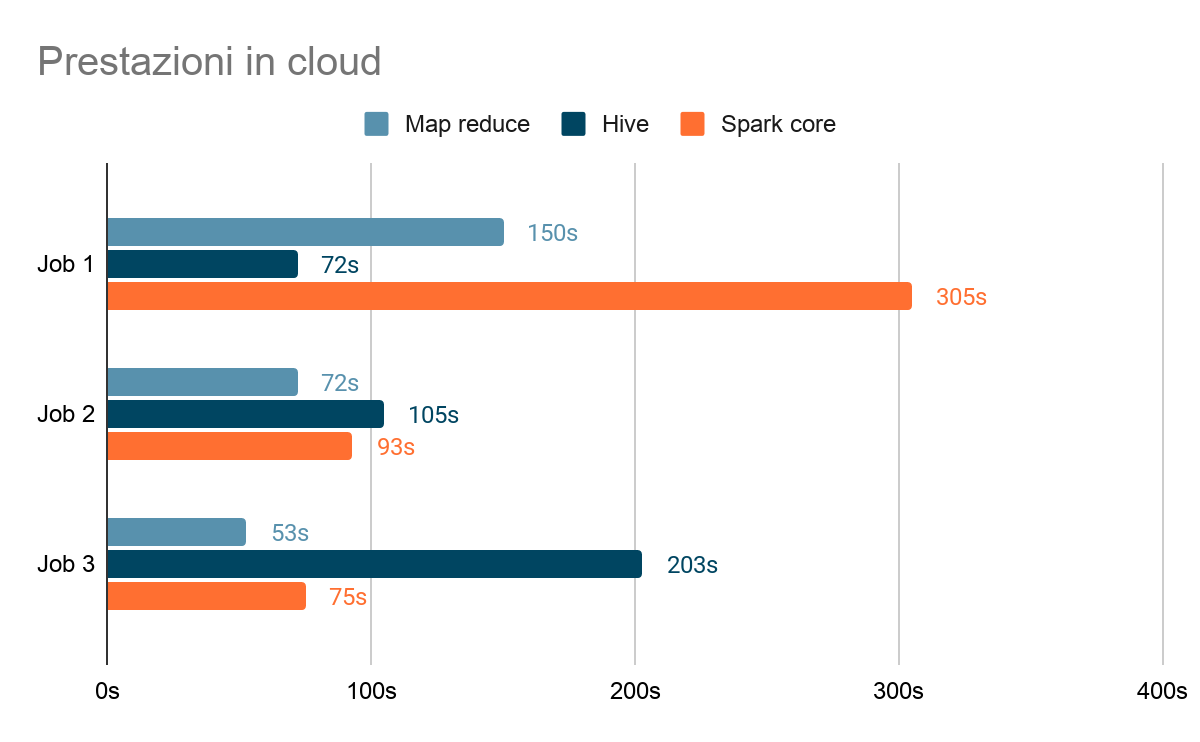
****

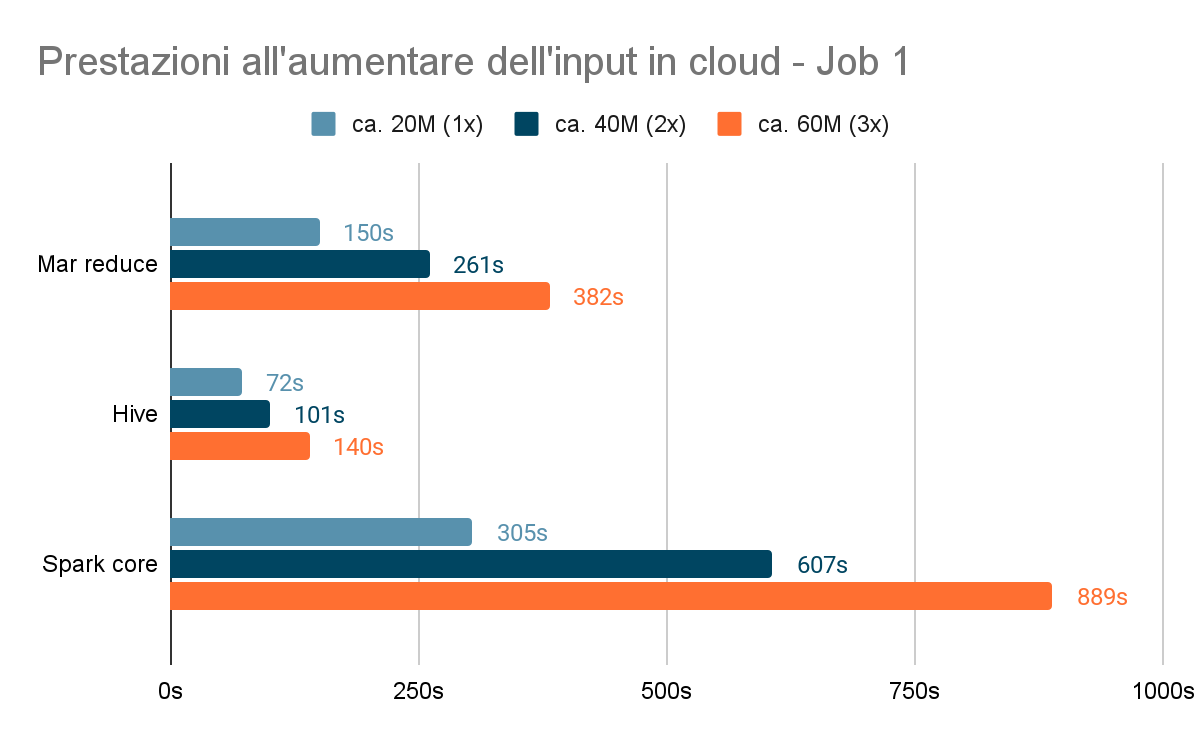
**7.2- Benchmark in cloud**

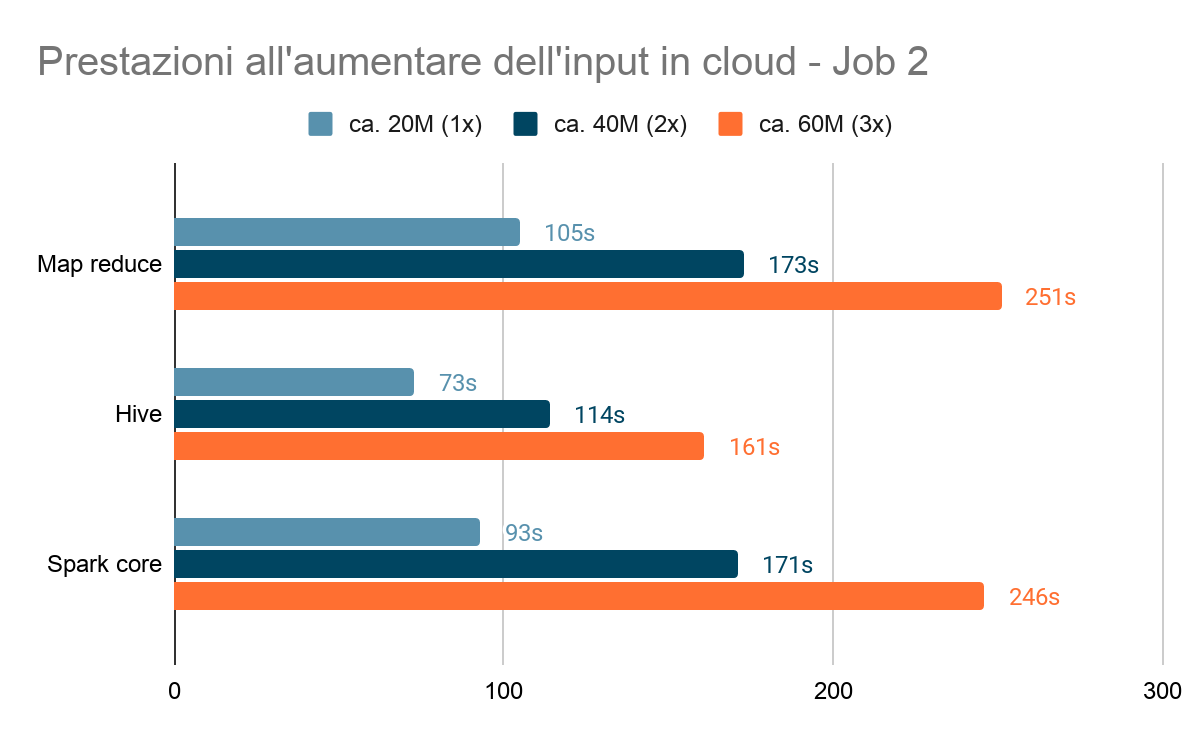
Tutti i test in cloud sono stati eseguiti su macchine EMR di tipo m5.xlarge.

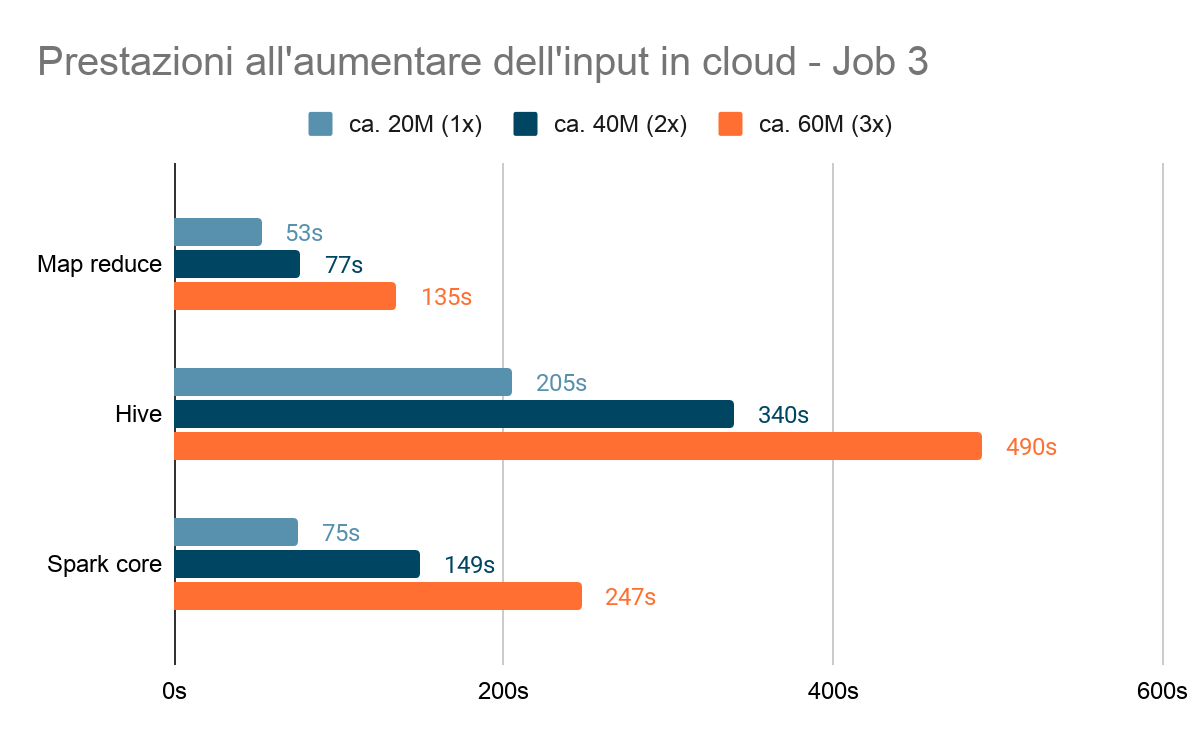
La configurazione scelta è stata emr-6.3.0, con un nodo master e 2 nodi slave, tuttavia questa configurazione è stata usata solo per Mapreduce e Spark core, in quanto Hive al momento dell’avvio impiegava svariati minuti (anche decine) solo per iniziare la prima operazione di mapping, sfalsando i risultati.

Non avendo trovato una soluzione rapida, si è scelto di testare Hive su un secondo cluster con configurazione software emr-5.33.0, che invece è esente da questa problematica.

****

****

****

****