Relazione Progetto Big Data 21/22

Claudio Vona, Beyza Özdemir

20 maggio 2022

Indice

Intruduzione

Questa relazione tratta del primo progetto del corso Big Data anno 21/22 di cui riportiamo ora la traccia:

Si consideri il dataset Amazon Fine Food Reviews di Kaggle1 , che contiene circa 500.000 recensioni di prodotti gastronomici rilasciati su Amazon dal 1999 al 2012. Il dataset è in formato CSV e ogni riga ha i seguenti campi:

- Id,
- ProductId (unique identifier for the product),
- UserId (unique identifier for the user),
- ProfileName,
- HelpfulnessNumerator (number of users who found the review helpful),
- HelpfulnessDenominator (number of users who graded the review),
- Score (rating between 1 and 5),
- Time (timestamp of the review expressed in Unix time),
- Summary (summary of the review),
- Text (text of the review).

A seguire i tre job richiesti riportando per ogni job anche le specifiche richieste.

1 Job 1

In questa sezione discuteremo di come abbiamo portato a termine il primo job utilizzando i tre programmi richiesti: Hadoop, Hive e Spark Core.

E' stato richiesto un job che sia in grado di generare, per ciascun anno, le dieci parole che sono state più usate nelle recensioni (campo text) in ordine di frequenza, indicando, per ogni parola, il numero di occorrenze della parola nell'anno.

1.1 Hadoop

Per svolgere il Job richiesto utilizzando Hadoop, è stato innanzitutto creato un mapper con il compito di estrarre da ogni riga del dataset, l'anno della recensione e il testo associato, infine è stato utilizzato un reducer per ottenere l'output richiesto.

Di seguito riportiamo lo pseudo codice di entrambi

```
MAPPER
SET our custom regex TO ",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)"
FOR  every line IN sys.stdin:

    SET line TO  stripped line
    SET rows TO an array from splitted line using our custom regex
    SET timestamp TO rows[7]
    SET text TO rows[9]
    IF timestamp != "Time":
        SET date TO extracted year from timestamp

       PRINT text and date our reducer

REDUCER
SET year_2_most_used_word TO {}
FOR every line IN sys.stdin:

    SET line TO  stripped line
    SET current_date, current_string TO our text and date from MAPPER
    SET words TO splitted words from current_string

    IF current_date not IN year_2_most_used_word:
        SET year_2_most_used_word[current_date] TO empty dictionary

    FOR every word IN words:
        IF current_date IN year_2_most_used_word and word IN year_2_most_used_word[current_date]:
            year_2_most_used_word[current_date][word] increment word counter
        ELSEIF current_date IN year_2_most_used_word:
            SET year_2_most_used_word[current_date][word] TO initialize counter with 0
FOR every year IN year_2_most_used_word:

    SET sorted_words TO sort associated set by year

    PRINT year and first 10 word with counter
```

1.1.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
1999: the, 12, is, 9, to, 8, and, 7, on, 5, a, 5, of, 4, in, 4, it, 4, this, 4
2000: the, 178, and, 102, to, 90, of, 87, a, 84, is, 65, that, 55, they, 39, film, 35, in, 34
2001: the, 64, to, 39, a, 34, and, 32, it, 18, is, 16, of, 15, in, 15, I, 15, was, 14
```

```
2002: the, 400, a, 310, and, 295, of, 211, is, 191, to, 178, in, 119, I, 111, this, 103, it, 94
2003: the, 497, a, 381, and, 343, to, 284, is, 265, of, 224, in, 198, I, 175, that, 128, for, 120
2004: the, 2462, a, 1674, and, 1609, of, 1262, to, 1242, is, 1085, I, 985, in, 837, it, 713,
for, 613
2005: the, 4903, and, 3468, a, 3257, to, 2603, of, 2584, I, 2349, is, 2304, in, 1743, it, 1546,
for, 1371
2006: the, 21042, and, 15709, a, 14625, I, 12564, to, 11762, of, 10673, is, 9447, it, 6747,
in, 6700, for, 6095
2007: the, 51758, and, 39753, a, 36745, I, 35780, to, 30941, of, 25546, is, 24194, it, 18430, in,
17253, for, 16788
2008: the, 93269, and, 70800, a, 66158, I, 64026, to, 55381, of, 46625, is, 41923, it, 33977,
for, 30018, in, 29440
```

1.2 Hive

Per svolgere il Job richiesto utilizzando Hive, si è andati ad eseguire una serie di query prima per sfoltire la tabella iniziale (ottenuta dal .csv) usando anche una custom trasform per estrarre l'anno dal timestamp e poi per ottenere il risultato aspettato. Di seguito riportiamo le query:

```
CREATE TABLE IF NOT EXISTS reviews (
id int,
productid string,
userId string,
nick string,
hn int,
hd int,
score int,
time int,
summary string,
text string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH '/home/cvona/Documents/Reviews.csv' overwrite INTO TABLE reviews;


SELECT * from reviews LIMIT 10;

ADD FILE /home/cvona/Documents/date_convert.py;

CREATE TABLE year2text AS
    SELECT TRANSFORM(reviews.text, reviews.time)
        USING 'python3 /home/cvona/Documents/date_convert.py' AS text, year
    FROM reviews;



CREATE TABLE exploded_text AS
    SELECT year, exp.word
    FROM  year2text LATERAL VIEW explode(split(regexp_replace(text,'[.!?\\-]',''),'\\s'))  exp AS wor
SELECT * FROM exploded_text LIMIT 10;


CREATE TABLE grouped_by_year AS
    SELECT year, word, count(word) AS occurence
    FROM exploded_text
    GROUP BY year, word;
```

```
CREATE TABLE year_word_sum AS

    SELECT year,  word, sum(occurence) AS somma
    FROM grouped_by_year
    GROUP BY year, word;


CREATE TABLE ten_most_used_word_per_year AS
    SELECT year, word, somma
    FROM (
        select *, ROW_NUMBER() OVER (PARTITION BY year ORDER BY year, somma DESC) AS rank
        from year_word_sum
    ) AS exp
    WHERE rank<=10;

    SELECT * from ten_most_used_word_per_year;
```

e la classe python:

```
    for line in sys.stdin:

    try:
        # remove whitespaces and trailing characters
        line = line.strip()

        # parse name and unix date using TAB as a separator
        text, time = line.split("\t")

        # try to convert the unix date to an integer
        try:
            time = int(time)
        except ValueError:
            continue

        # build a datetime object from the unix time
        datetime_obj = datetime.utcfromtimestamp(time)

        # get the output date string
        year = datetime.fromtimestamp(int(time)).strftime('%Y')

        # print output items to stdout, using TAB as a separator
        print("\t".join([text, year]))
    except:
        import sys
        print(sys.exc_info())
```

1.2.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
1999: the, 12, is, 9, to, 8, and, 7, on, 5, a, 5, of, 4, in, 4, it, 4, this, 4
2000: the, 178, and, 102, to, 90, of, 87, a, 84, is, 65, that, 55, they, 39, film, 35, in, 34
2001: the, 64, to, 39, a, 34, and, 32, it, 18, is, 16, of, 15, in, 15, I, 15, was, 14
2002: the, 400, a, 310, and, 295, of, 211, is, 191, to, 178, in, 119, I, 111, this, 103, it, 94
2003: the, 497, a, 381, and, 343, to, 284, is, 265, of, 224, in, 198, I, 175, that, 128, for, 120
2004: the, 2462, a, 1674, and, 1609, of, 1262, to, 1242, is, 1085, I, 985, in, 837, it,
```

```
713, for, 613
2005: the, 4903, and, 3468, a, 3257, to, 2603, of, 2584, I, 2349, is, 2304, in, 1743, it,
1546, /><br, 1371
2006: the, 21042, and, 15709, a, 14625, I, 12564, to, 11762, of, 10673, is, 9447, it, 6747,
in, 6700, for, 6095
2007: the, 51758, and, 39753, a, 36745, I, 35780, to, 30941, of, 25546, is, 24194, it, 18430,
in, 17253, for, 16788
2008: the, 93269, and, 70800, a, 66158, I, 64026, to, 55381, of, 46625, is, 41923,
it, 33977, for, 30018, in, 29440
```

1.3 Spark Core

Per svolgere il Job richiesto utilizzando Spark Core, si è sfruttato l'utilizzo degli RDD, dopo aver trasformato il file .csv in un RDD, siamo andati a svolgere una serie di trasformazioni per ottenere il risultato voluto.

```
lines = spark.sparkContext.textFile(input_filepath).cache()

filtered_lines = lines.filter(word: ! word.startswith("Id") & ! word.endswith("Text"))

stripped_lines = filtered_lines.map(strip())
years_text = stripped_lines.map( line: (datetime.fromtimestamp(int(re.split(regex, line)))

years_text_reduced = years_text.reduceByKey(a, b: a + " " + b)

years_word = years_text_reduced.map((item[0], findall('[^,;\\s]+', item[1])))

years_most_used_words = years_word.map( mostcommon(item[0], Counter(item[1])

years_top_10_most_used_words = years_most_used_words.map(item(item[0], item[1..10]))

saveAsTextField(years_top_10_most_used_words)
```

1.3.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
1999: the, 12, is, 9, to, 8, and, 7, on, 5, a, 5, of, 4, in, 4, it, 4, this, 4
2000: the, 178, and, 102, to, 90, of, 87, a, 84, is, 65, that, 55, they, 39, film, 35, in, 34
2001: the, 64, to, 39, a, 34, and, 32, it, 18, is, 16, of, 15, in, 15, I, 15, was, 14
2002: the, 400, a, 310, and, 295, of, 211, is, 191, to, 178, in, 119, I, 111, this, 103, it, 94
2003: the, 497, a, 381, and, 343, to, 284, is, 265, of, 224, in, 198, I, 175, that, 128, for, 120
2004: the, 2462, a, 1674, and, 1609, of, 1262, to, 1242, is, 1085, I, 985, in, 837, it, 713,
for, 613
2005: the, 4903, and, 3468, a, 3257, to, 2603, of, 2584, I, 2349, is, 2304, in, 1743, it, 1546,
for, 1371
2006: the, 21042, and, 15709, a, 14625, I, 12564, to, 11762, of, 10673, is, 9447, it, 6747,
in, 6700, for, 6095
2007: the, 51758, and, 39753, a, 36745, I, 35780, to, 30941, of, 25546, is, 24194, it, 18430, in,
17253, for, 16788
2008: the, 93269, and, 70800, a, 66158, I, 64026, to, 55381, of, 46625, is, 41923, it, 33977,
for, 30018, in, 29440
```

1.4 Grafici e complessità

Per quanto riguarda la complessità del codice, gli algoritmi da noi progettati mostrano una tendenza ad una complessità di $\theta(n)$, come dimostra il grafico sottostante.

Da evidenziare che i test sono stati eseguiti raddoppiando ogni volta il dataset e relativi record fonirti.

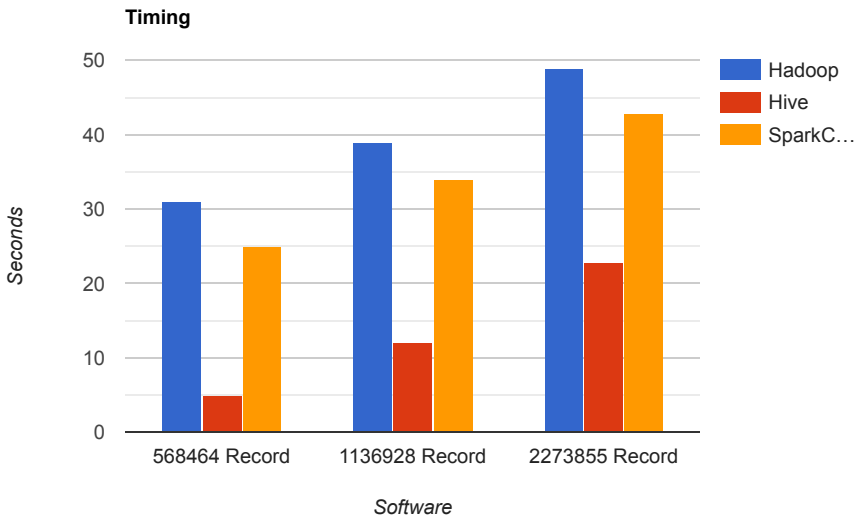

Figura 1: Grafico che ci mostra l'andamento al aumentare dei records.

2 Job 2

In questa sezione discuteremo di come abbiamo portato a termine il secondo job utilizzando i tre programmi richiesti: Hadoop, Hive e Spark Core.

E' stato chiesto un job che sia in grado di generare, per ciascun utente, i prodotti preferiti (ovvero quelli che ha recensito con il punteggio più alto) fino a un massimo di 5,indicando ProductId e Score. Il risultato deve essere ordinato in base allo UserId.

2.1 Hadoop

Per svolgere il Job richiesto utilizzando Hadoop, è stato innanzitutto creato un mapper con il compito di estrarre da ogni riga del dataset, l'anno della recensione e il testo associato, infine è stato utilizzato un reducer per ottenere l'output richiesto.

Di seguito riportiamo lo pseudo codice di entrambi

```
MAPPER
SET our custom regex TO ",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)"
FOR  every line IN sys.stdin:

    SET line TO  stripped line
    SET rows TO an array from splitted line using our custom regex
    SET productid = rows[1]
    SET userid = rows[2]
    SET score = rows[6]
    IF score != "Score":
       PRINT userid,productid, and score for our reducer
```

```
REDUCER
SET userID_2_products TO {}
FOR every line IN sys.stdin:

    SET line TO  stripped line
    SET current_user, current_product, score TO userid,productid, and score from our mapper
    IF current_user NOT IN userID_2_products:
        userID_2_products[current_user] = empty set
    IF current_user in userID_2_products:
        userID_2_products[current_user] append (pair productid, score)

FOR every userid in userID_2_products:
    SET product_2_score_list TO userID_2_products[user]
    SET ordered_list TO sorted product_2_score_list by score
    PRINT userid and top 5 item from ordered_list
```

2.1.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
#oc-R103C0QSV1DF5E [('B006Q820X0', '5')]
#oc-R109MU5OBBZ59U [('B008I1XPKA', '5')]
#oc-R10LFEMQEW6QGZ [('B008I1XPKA', '5')]
#oc-R10LT57ZGIB140 [('B0026LJ3EA', '3')]
#oc-R10UAO29WVWIUI [('B006Q820X0', '1')]
#oc-R115TNMSPFT9I7 [('B005ZBZLT4', '2'), ('B007Y59HVM', '2')]
#oc-R119LM8D59ZW8Y [('B005DVVB9K', '1')]
#oc-R11D9D7SHXIJB9 [('B005HG9ERW', '5'), ('B005HG9ET0', '5'), ('B005HG9ESG', '5')]
#oc-R11D9LKDAN5NQJ [('B008I1XPKA', '3')]
#oc-R11DNU2NBKQ23Z [('B007Y59HVM', '1'), ('B005ZBZLT4', '1')]
#oc-R11O5J5ZVQE25C [('B005HG9ERW', '5'), ('B005HG9ET0', '5'), ('B005HG9ESG', '5')]
#oc-R11PW3CFBB4BEP [('B006Q820X0', '3')]
#oc-R11T1PHWNO7KEZ [('B006Q820X0', '3')]
#oc-R120LO6OLNDPCG [('B006Q820X0', '1')]
```

2.2 Hive

Per svolgere il Job richiesto utilizzando Hive, si è andati ad eseguire una serie di query prima per sfoltire la tabella iniziale (ottenuta dal .csv) e poi per ottenere il risultato aspettato. Di seguito riportiamo le query:

```
DROP TABLE reviews;
DROP TABLE most_scored_product_for_user;


CREATE TABLE IF NOT EXISTS reviews (
id int,
productid string,
userid string,
nick string,
hn int,
hd int,
score int,
time int,
summary string,
text string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '/home/cvona/Documents/Reviews.csv' overwrite INTO TABLE reviews;


SELECT * from reviews LIMIT 10;

CREATE TABLE most_scored_product_for_user as
    SELECT userid, productid, score
    FROM (
     SELECT *, ROW_NUMBER() OVER (PARTITION BY userid ORDER BY userid, score DESC) AS rank
     FROM reviews
    ) AS exp
    WHERE rank <= 5;
```

2.2.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
#oc-R103C0QSV1DF5E B006Q820X0 5
#oc-R103C0QSV1DF5E B006Q820X0 5
#oc-R10LFEMQEW6QGZ B008I1XPKA 5
#oc-R10LT57ZGIB140 B0026LJ3EA 3
#oc-R115TNMSPFT9I7 B007Y59HVM 2
#oc-R115TNMSPFT9I7 B005ZBZLT4 2
#oc-R119LM8D59ZW8Y B005DVVB9K 1
#oc-R11DNU2NBKQ23Z B005ZBZLT4 1
#oc-R11DNU2NBKQ23Z B007Y59HVM 1
#oc-R11PW3CFBB4BEP B006Q820X0 3
#oc-R11T1PHWNO7KEZ B006Q820X0 3
#oc-R12KPBODL2B5ZD B007OSBE1U 1
```

2.3 Spark Core

Per svolgere il Job richiesto utilizzando Spark Core, si è sfruttato l'utilizzo degli RDD, dopo aver trasformato il file .csv in un RDD, siamo andati a svolgere una serie di trasformazioni per ottenere il risultato voluto.

```
lines = spark.sparkContext.textFile(input_filepath).cache()

filtered_lines = lines.filter(word: ! word.startswith("Id") & ! word.endswith("Text"))

user_product= filtered_lines.map(line: (re.split[(regex, line)[2], re.split(regex, line)[1], re.split

user_product_reduced = user_product.reduceByKey(a, b: a + b)

user_products = user_product_reduced.map(item: (item[0], item[1])). map(item: (item[0], [(x.[0], x.[1

ordered_RDD = user_2_products_RDD.map(f=lambda x: (x[0], sorted(x[1], key=lambda item: item[1], rever

top_five = ordered.map(item: (item[0], item[1 .. 5])).coalesce(1)

to_order = top_five.collect()

final = (sorted(to_order, x: x[0])).coalesce(1)

saveAsTextFile(final)
```

2.3.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
('#oc-R103C0QSV1DF5E', [('B006Q820X0', '5')])
('#oc-R109MU5OBBZ59U', [('B008I1XPKA', '5')])
('#oc-R10LFEMQEW6QGZ', [('B008I1XPKA', '5')])
('#oc-R10LT57ZGIB140', [('B0026LJ3EA', '3')])
('#oc-R10UA029WVWIUI', [('B006Q820X0', '1')])
('#oc-R115TNMSPFT9I7', [('B005ZBZLT4', '2'), ('B007Y59HVM', '2')])
('#oc-R119LM8D59ZW8Y', [('B005DVVB9K', '1')])
('#oc-R11D9D7SHXIJB9', [('B005HG9ESG', '5'), ('B005HG9ERW', '5'), ('B005HG9ET0', '5')])
('#oc-R11D9LKDAN5NQJ', [('B008I1XPKA', '3')])
('#oc-R11DNU2NBKQ23Z', [('B005ZBZLT4', '1'), ('B007Y59HVM', '1')])
```

2.4 Grafici e complessità

Per quanto riguarda la complessità del codice, gli algoritmi da noi progettati mostrano una tendenza ad una complessità di $\theta(n)$, come dimostra il grafico sottostante.

Da evidenziare che i test sono stati eseguiti raddoppiando ogni volta il dataset e relativi record fonirti.

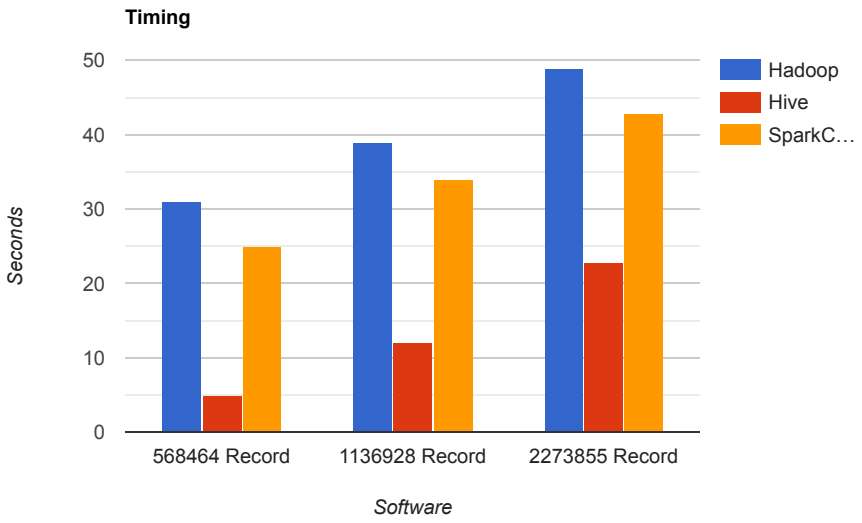

Figura 2: Grafico che ci mostra l'andamento al aumentare dei records.

3 Job 3

In questa sezione discuteremo di come abbiamo portato a termine il terzo job utilizzando i tre programmi richiesti: Hadoop, Hive e Spark Core.

E' stato chiesto un job in grado di generare coppie di utenti con gusti affini, dove due utenti hanno gusti affini se hanno recensito con score superiore o uguale a 4 almeno tre prodotti in comune, indicando le coppie di utenti e i prodotti recensiti che condividono. Il risultato deve essere ordinato in base allo UserId del primo elemento della coppia e non deve presentare duplicati.

3.1 Hadoop

Per svolgere il Job richiesto utilizzando Hadoop, è stato innanzitutto creato un mapper con il compito di estrarre da ogni riga del dataset, l'anno della recensione e il testo associato e infine è stato utilizzato un reducer per ottenere l'output richiesto.

Di seguito riportiamo lo pseudo codice di entrambi

```
MAPPER
SET regex TO ",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)"
FOR line IN sys.stdin:

    SET line TO line.strip()
    SET rows TO re.split(regex, line)

    SET score TO rows[6]
    If score >=4:
        SET userID TO rows[2]
        SET productID TO rows[1]
```

```
        SET userID and productID for the reducer

REDUCER
SET userID_2_products TO {}
SET products_2_userID TO {}
SET filtered_userID_2_products TO {}
SET users_2_shared_products TO {}

FOR every line IN sys.stdin:

    line TO stripped line
    current_user, current_product TO userID and productID from our mapper

    IF current_user not IN userID_2_products:
        userID_2_products[current_user] TO set()
    IF current_user IN userID_2_products:
        userID_2_products[current_user] add current_product to set

    IF current_product not IN products_2_userID:
        products_2_userID[current_product] TO set()
    IF current_product IN products_2_userID:
        products_2_userID[current_product] add current_user to set

FOR every product IN products_2_userID.keys():
    pairs TO create a list of pair of every  unique combination of the userID

    FOR every pair IN pairs:
        IF pair and tuple(reversed(pair)) not IN users_2_shared_products:
            products_intersection TO intersection of product list of the users
            IF len(products_intersection) >= 3:
                users_2_shared_products[pair] TO products_intersection

OrderedKeys TO  users_2_shared_products keys sorted by user id
FOR every pair IN OrderedKeys:
    sharedProducts TO users_2_shared_products[pair]
    OUTPUT("%s\t%s" % (pair, sharedProducts))
```

3.1.1 Output

Utilizzando il dataset fornito, ecco le prime 10 righe ottenute dell'output

```
('#oc-R11D9D7SHXIJB9', '#oc-R162D7S0A880MV') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R19EJ3VEA88T6O') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R1K4OCJ8HEIEDY') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R1QHGBT11WAS7G') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R1UO6NAAGVBW7Z') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R2M17G7NB9RAIV') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R2W66Y63G88976') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R2XZVYL146WRFL') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R2YPVWM76O2TFX') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
('#oc-R11D9D7SHXIJB9', '#oc-R31AI08Q9HLVF1') {'B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW'}
```

3.2 Hive

Per svolgere il Job richiesto utilizzando Hive, si è andati ad eseguire una serie di query prima per sfoltire la tabella iniziale (ottenuta dal .csv) e poi per ottenere il risultato aspettato. Di seguito riportiamo le query:

```
CREATE TABLE IF NOT EXISTS reviews (
id int,
productid string,
userid string,
nick string,
hn int,
hd int,
score int,
time int,
summary string,
text string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH '/home/cvona/Documents/Reviews.csv' overwrite INTO TABLE reviews;

CREATE TABLE score_greater_than_3 AS
SELECT userid, productid, score
FROM reviews
WHERE score >= 4;

CREATE TABLE result AS
SELECT s1.userid as userid1, s2.userid as userid2,
concat_ws(',',collect_list(s1.productid))
FROM score_greater_than_3 s1 inner join score_greater_than_3 s2 on s1.productid=s2.productid and s1.u
GROUP BY s1.userid, s2.userid
HAVING count(*)>=3;

INSERT OVERWRITE DIRECTORY  '/user/cvona/output/dir' row format delimited fields terminated by '\t' s
```

3.2.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
#oc-R11D9D7SHXIJB9 #oc-R163CP16SRRI50B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 AY6A8KPYCE6B0 B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 AW7BIYHXUIZ62 B005HG9ESG,B005HG9ERW,B005HG9ET0
#oc-R11D9D7SHXIJB9 ASB4QD6YZJ7EX B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 APP35M28G2U51 B005HG9ESG,B005HG9ERW,B005HG9ET0
#oc-R11D9D7SHXIJB9 AGEKVD8JPZQMT B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 A3VBXQKRM7A4JR B005HG9ESG,B005HG9ERW,B005HG9ET0
#oc-R11D9D7SHXIJB9 A3MUSWDCTZINQZ B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 A3KDO3XV0MK1GX B005HG9ESG,B005HG9ERW,B005HG9ET0
#oc-R11D9D7SHXIJB9 A3K91X9X2ARDOK B005HG9ET0,B005HG9ERW,B005HG9ESG
#oc-R11D9D7SHXIJB9 A3BKNXX8QFIXIV B005HG9ESG,B005HG9ERW,B005HG9ET0
```

3.3 Spark Core

Per svolgere il Job richiesto utilizzando Spark Core, si è sfruttato l'utilizzo degli RDD, dopo aver trasformato il file .csv in un RDD, siamo andati a svolgere una serie di trasformazioni per ottenere il risultato voluto.

```
lines = spark.sparkContext.textFile(input_filepath).cache()
filtered_lines = lines.filter(word: ! word.startswith("Id") & ! word.endswith("Text"))
more_than_4_product_score = filtered_lines.filter(line: re.split(regex, line)[6] >= 4)
product_userID = more_than_4_product_score.
\map(line: (re.split(regex, line)[1], re.split(regex, line)[2]))
userID_product = more_than_4_product_score.
```

```
\map(line: (re.split(regex, line)[2], re.split(regex, line)[1]))
product_userID_reduced = product_userID.reduceByKey(a, b: a + " " + b)
userID_product_reduced = userID_product.reduceByKey(a, b: a + " " + b)
filtered_product_userID_reduced = product_userID_reduced.filter(x: len(x[1].split(" ")) > 1).
\ map(x: (x[0], set(x[1].split(" "))))
filtered_userID_product_reduced = userID_product_reduced.map(x: (x[0], set(x[1].split(" "))))
product_userID_dictionary = filtered_product_userID_reduced.collectAsMap()
userID_product_dictionary = filtered_userID_product_reduced.collectAsMap()
final_dict = {}
for product in product_2_userID_dictionary.keys(){
pairs = list(itertools.combinations(product_2_userID_dictionary[product], 2))
for pair in pairs{
        if pair & tuple(reversed(pair)) ! in final_dict{
            products_intersection = userID_product_dictionary[pair[0]]. \
                intersection(userID_product_dictionary[pair[1]])}
            if len(products_intersection) >= 3{
                final_dict[pair] = products_intersection}
    }
 }
final = sorted(final_dict.items(), x: (x[0], x[1]))
collapsed = final.coalesce(1)
saveAsTextFile(collapsed_)
```

3.3.1 Output

Utilizzando il dataset fornito, ecco le prime dieci righe dell'output ottenuto

```
(('#oc-R11D9D7SHXIJB9', '#oc-R19EJ3VEA88T6O'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R1GSBW9QIVY489'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R1QHGBT11WAS7G'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R1VRD09DW4H2HI'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R2HWL8UHAIMFRS'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R2K9AJ2LWO7ZUJ'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R2W66Y63G88976'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R2XZVYL146WRFL'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R3DERHJ8UWPZZ'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
(('#oc-R11D9D7SHXIJB9', '#oc-R3OS88C8I7GSS5'), {'B005HG9ET0', 'B005HG9ERW', 'B005HG9ESG'})
```

3.4 Grafici e complessità

Per quanto riguarda la complessità del codice, gli algoritmi da noi progettati mostrano una tendenza ad una complessità di $\theta(n)$, come dimostra il grafico sottostante.

Da evidenziare che i test sono stati eseguiti raddoppiando ogni volta il dataset e relativi record fonirti.

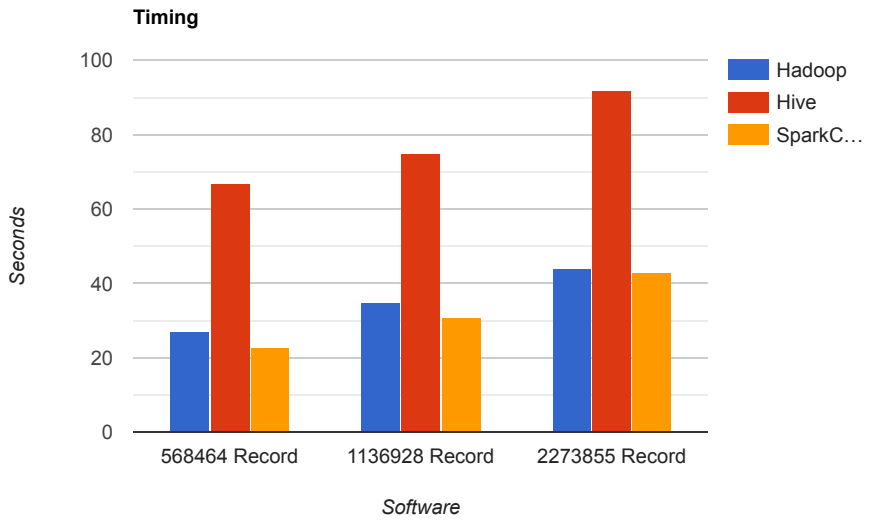

Figura 3: Grafico che ci mostra l'andamento al aumentare dei records.

3.5 Cluster

Abbiamo provato anche ad eseguire il Job 3 su 3 cluster online giovando di un boost di prestazioni non indifferente

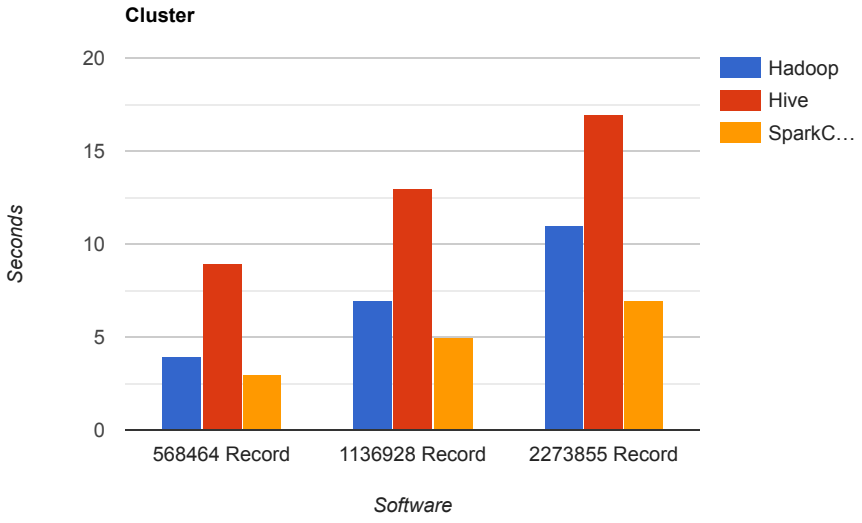

Figura 4: Grafico che ci mostra l'andamento al aumentare dei records.