

Primo progetto Big Data

Manuel Granchelli

Matricola: 512406

*Dipartimento di Ingegneria Informatica
Università degli Studi Roma Tre*

man.granchelli@stud.uniroma3.it

Repo GitHub: <https://github.com/mgranchelli/Progetto1BigData>

Introduzione

In questo rapporto verranno illustrate le procedure di analisi ed elaborazione di un dataset attraverso l'utilizzo di tre diverse tecnologie. Il dataset utilizzato è il seguente: **Amazon Fine Food Reviews** e le tecnologie utilizzate le seguenti: **MapReduce**, **Hive** e **Spark**. Il dataset, in formato csv, contiene circa 500.000 recensioni di prodotti gastronomici rilasciati su Amazon dal 1999 al 2012 e ogni riga del dataset contiene 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).

Sono stati progettati e realizzati per ciascuna tecnologia tre diversi Job:

1. 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.
2. 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.
3. 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.

Gli script, i notebook in Python utilizzati, i file di output e tutto il restante materiale che verrà citato, ad eccezione dei dataset, sono presenti anche all'interno del **Repo GitHub**. I file log inseriti nel seguente documento sono relativi all'output del terminale e gli output sono relativi ad un numero limitato di righe dei file di output. Per mancanza di tempo non è stato possibile testare i vari Job su cluster.

Indice

| | | |
|----------|-------------------------------|-----------|
| 1 | Analisi del dataset | 1 |
| 2 | Job 1 | 1 |
| 2.1 | MapReduce | 1 |
| 2.1.1 | Pseudocodice | 2 |
| 2.1.2 | Script | 2 |
| 2.1.3 | Output | 2 |
| 2.1.4 | Log | 3 |
| 2.2 | Hive | 3 |
| 2.2.1 | Script | 3 |
| 2.2.2 | Output | 4 |
| 2.2.3 | Log | 4 |
| 2.3 | Spark | 4 |
| 2.3.1 | Pseudocodice | 5 |
| 2.3.2 | Script | 5 |
| 2.3.3 | Output | 5 |
| 2.3.4 | Log | 5 |
| 2.4 | Tempi di esecuzione | 6 |
| 3 | Job 2 | 6 |
| 3.1 | MapReduce | 6 |
| 3.1.1 | Pseudocodice | 6 |
| 3.1.2 | Script | 7 |
| 3.1.3 | Output | 7 |
| 3.1.4 | Log | 8 |
| 3.2 | Hive | 8 |
| 3.2.1 | Script | 8 |
| 3.2.2 | Output | 8 |
| 3.2.3 | Log | 9 |
| 3.3 | Spark | 9 |
| 3.3.1 | Pseudocodice | 9 |
| 3.3.2 | Script | 9 |
| 3.3.3 | Output | 10 |
| 3.3.4 | Log | 10 |
| 3.4 | Tempi di esecuzione | 11 |
| 4 | Job 3 | 11 |
| 4.1 | MapReduce | 11 |
| 4.1.1 | Pseudocodice | 12 |
| 4.1.2 | Script | 12 |
| 4.1.3 | Output | 13 |
| 4.1.4 | Log | 13 |
| 4.2 | Hive | 13 |
| 4.2.1 | Script | 14 |
| 4.2.2 | Output | 14 |
| 4.2.3 | Log | 14 |
| 4.3 | Spark | 15 |
| 4.4 | Pseudocodice | 15 |
| 4.4.1 | Script | 15 |
| 4.4.2 | Output | 15 |
| 4.4.3 | Log | 16 |
| 4.5 | Tempi di esecuzione | 16 |

1 Analisi del dataset

L'analisi del dataset è stata svolta all'interno del notebook *Data analysis.ipynb* utilizzando Python. Il file csv del dataset è stato caricato in un DataFrame all'interno del notebook, utilizzando la libreria pandas. Per prima cosa è stata verificata la presenza di dati nulli. Sono presenti in piccola parte nelle colonne *ProfileName* e *Summary*. In base ai Job da svolgere, le due colonne non sono utili al fine dello svolgimento dei diversi Job. Analizzando il testo dei vari Job è possibile osservare che le colonne necessarie per svolgere il Job 1 sono: *Time* e *Text* e per i Jobs 3 e 4 sono: *ProductId*, *UserId* e *Score*.

Per questo motivo sono stati creati due DataFrame, uno per il Job 1 contenente solamente le due colonne citate e l'altro per i Jobs 2 e 3 contenente solamente le altre tre colonne sopra citate. Inoltre, analizzando nel dettaglio la colonna *Text* è stato osservato che nei campi sono presenti segni di punteggiatura e tag HTML, i quali sono stati rimossi durante la progettazione del Job 1 con le diverse tecnologie.

Infine, per poter generare file di diverse dimensioni, sempre all'interno del file *Data analysis.ipynb* è presente una funzione che genera, a partire dai due DataFrame, tre file di diversa dimensione per ciascun DataFrame. Le dimensioni dei file sono state impostate nel seguente modo: (a) metà dataset, (b) dataset stesso e (c) due volte il dataset. In totale, quindi, la funzione restituisce tre file csv che sono stati utilizzati nel Job 1 e tre file csv che sono stati utilizzati nei Jobs 2 e 3.

Gli output completi dei vari Jobs con le diverse tecnologie sono presenti sul **Repo GitHub**, mancano gli output del Job 3 per il file (c) a causa delle dimensioni elevate (superiori a 100MB) dei file di output.

2 Job 1

L'obiettivo del seguente Job è quello di restituire per ciascuno anno le dieci parole più usate nel campo text in ordine di frequenza, indicando, per ogni parola in numero di occorrenze nell'anno. Nel seguente Job sono stati utilizzati i tre file di diversa dimensione generati precedentemente tutti aventi i seguenti due campi: *Time* e *Text*.

2.1 MapReduce

L'implementazione MapReduce è stata svolta creando due script, uno script mapper ed uno reducer. Il mapper legge il file csv una riga per volta ed esegue uno split della riga in due parti. Ciascuna riga contiene il valore time nel formato Unix time, che viene convertito per ottenere l'anno e il testo che viene ripulito da eventuali segni di punteggiatura, spazi in eccesso e tag HTML. Il testo successivamente è stato suddiviso in parole.

Il mapper per ciascun anno e parola stampa una stringa contenente l'anno, la parola e l'intero 1. L'intero 1 viene utilizzato nel file reducer per contare il numero di occorrenze. Infatti, lo script reducer, prende in input le stringhe stampate dal mapper e tramite un dizionario conta il numero di occorrenze della parola nell'anno. Il reducer ordina per ogni anno le parole in base al numero di occorrenze e stampa in output per ogni anno le 10 parole più usate.

2.1.1 Pseudocodice

Algorithm Mapper Job 1

```
1: for line in stdin do
2:   line ← line.strip()
3:   year, words ← line.split()
4:   Remove tag HTML inside words, punctuation and extra whitespace
5:   words ← words.split()
6: end for
7: for word in words do
8:   Output (year, word.lower(), 1)
9: end for
```

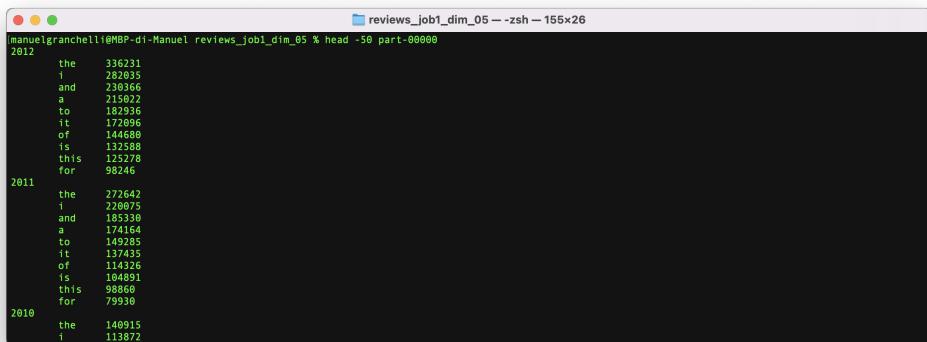
Algorithm Reducer Job 1

```
1: Initialize year_words dictionary
2: for line in stdin do
3:   year, current_word, current_count ← line.split()
4:   if year not in year_words then
5:     Initialize year_words[year] dictionary
6:   end if
7:   if current_word not in year_words[year] then
8:     year_words[year][current word] ← 0
9:   end if
10:  year_words[year][current word] += current_count
11: end for
12: for year, words in sort(year_words) do
13:   Output (year)
14:   for word in sort(words) do
15:     Output (word, count)
16:   end for
17: end for
```

2.1.2 Script

I due script utilizzati sono *mapper.py* e *reducer.py* presenti all'interno del **Repo GitHub**.

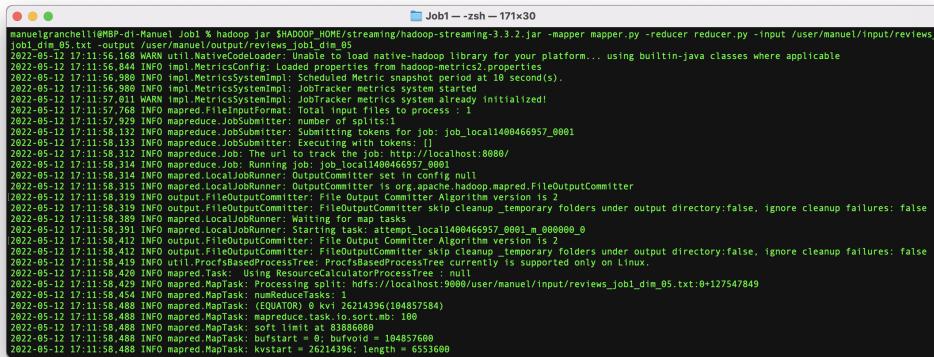
2.1.3 Output



```
manuelgranchelli@MBP-di-Manuel reviews_job1_dim_05 % head -50 part-00000
2012
the 336231
i 282035
and 238366
a 215022
to 182936
it 172096
of 144688
is 125288
this 123278
for 98246
2011
the 272642
i 220075
and 185318
a 174154
to 149285
it 137435
of 114326
is 104891
this 98860
for 79938
2010
the 140915
i 113872
```

Ouput Job 1 MapReduce - File (a)

2.1.4 Log



```
Job1 -- zsh -- 17x30
manuelfranchelli@MBP-d-Manuel:~/Manuel$ hadoop jar $HADOOP_HOME/streaming/hadoop-streaming-3.3.2.jar -mapper mapper.py -reducer reducer.py -input /user/manuel/input/reviews_05.txt -output /user/manuel/output/reviews_job1.dim_05
2022-05-12 17:11:56.168 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2022-05-12 17:11:56.844 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2022-05-12 17:11:56.988 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2022-05-12 17:11:57.001 INFO impl.MetricsSystemImpl: JobMetrics2 metrics system started
2022-05-12 17:11:57.911 WARN impl.MetricsSystemImpl: JobMetrics2 metrics system may not have initialized!
2022-05-12 17:11:57.768 INFO mapred.FileInputFormat: Total input files to process : 1
2022-05-12 17:11:57.929 INFO mapreduce.JobSubmitter: number of splits:1
2022-05-12 17:11:58.039 INFO mapreduce.JobSubmitter: Submitting application master job: job_local1408466957_0001
2022-05-12 17:11:58.132 INFO mapreduce.Job: Number of splits: 1
2022-05-12 17:11:58.312 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2022-05-12 17:11:58.314 INFO mapreduce.Job: Running job: job_local1408466957_0001
2022-05-12 17:11:58.314 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2022-05-12 17:11:58.314 INFO mapred.LocalJobRunner: OutputCommitter org.apache.hadoop.mapred.FileOutputCommitter
2022-05-12 17:11:58.319 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-05-12 17:11:58.319 INFO output.FileOutputCommitter: skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-05-12 17:11:58.389 INFO mapred.LocalJobRunner: Waiting for map tasks
2022-05-12 17:11:58.391 INFO mapred.LocalJobRunner: Starting task: attempt_local1408466957_0001_m_000000_0
2022-05-12 17:11:58.412 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
2022-05-12 17:11:58.412 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-05-12 17:11:58.419 INFO util.ProcessBasedTree: ProcsBasedProcessTree currently is supported only on Linux.
2022-05-12 17:11:58.428 INFO mapred.Task: Using ResourceCalculatorProcessTree : null
2022-05-12 17:11:58.428 INFO mapred.Task: Task ID: attempt_local1408466957_0001_m_000000_0
2022-05-12 17:11:58.448 INFO mapred.MapTask: numReduceTasks : 1
2022-05-12 17:11:58.448 INFO mapred.MapTask: (EQUATOR) 0 kv1 26214396(16485784)
2022-05-12 17:11:58.448 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2022-05-12 17:11:58.448 INFO mapred.MapTask: mapreduce.task.io.sort.size.per.line: 8388608
2022-05-12 17:11:58.448 INFO mapred.MapTask: mapreduce.task.io.sort.spill.size: 104857600
2022-05-12 17:11:58.448 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
```

Log Job 1 MapReduce - File (a)

2.2 Hive

L'implementazione Hive è stata svolta creando una tabella con due campi *time* e *text* nella quale è stato caricato il file csv. Successivamente tramite un *SELECT TRANSFORM* e uno script scritto in Python è stata creata una nuova tabella contenente in ogni riga l'anno e la parola. Lo script Python utilizzato è simile al mapper utilizzato nell'implementazione MapReduce, l'unica differenza è in fase di output dove in questo caso lo script restituisce l'anno e la parola senza l'intero 1. Lo script prende in input il campo *time* e *text* della tabella nella quale viene caricato il file csv ed è stato utilizzato principalmente per poter eseguire, come nel caso precedente, una pulizia del campo *text* e per ottenere l'anno dal campo *time*.

Infine, è stata creata la tabella di output con tre campi (*year*, *word*, *number*), i quali conterranno l'anno, la parola e il numero di occorrenze della parola nell'anno. La tabella di output è stata creata ordinandola in base all'anno e per ogni anno ci saranno le dieci parole con il numero di occorrenze più elevato, anch'esse ordinate in base alle occorrenze.

2.2.1 Script

Lo script Python utilizzato è il seguente: *year_words.py* e lo script hql utilizzato è il seguente: *job1.hql*. Entrambi sono presenti all'interno del **Repo GitHub**.

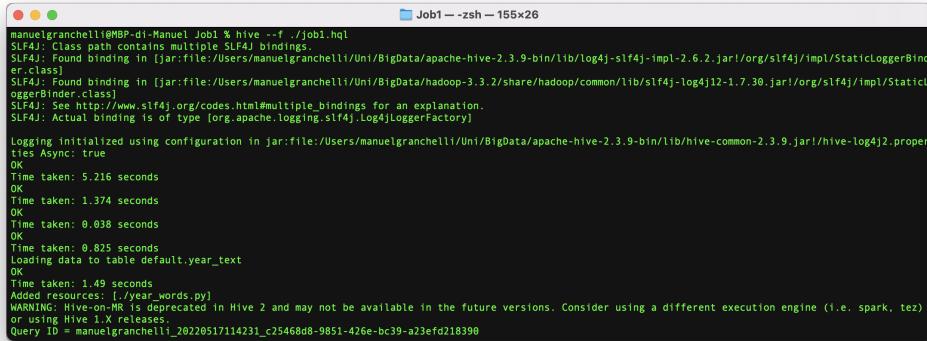
2.2.2 Output



```
OK
2012 the 336231
2012 i 282035
2012 and 259356
2012 to 215927
2012 it 182936
2012 of 172096
2012 is 132588
2012 this 125278
2012 for 125146
2011 the 272642
2011 i 220075
2011 and 185338
2011 a 174164
2011 to 149285
2011 it 137535
2011 of 114326
2011 is 104891
2011 this 98860
2011 for 79930
2010 the 140915
2010 i 113872
2010 and 90965
2010 a 98874
2010 to 75910
```

Ouput Job 1 Hive - File (a)

2.2.3 Log



```
manuelgranchelli@MBP-d1-Manuel:~/Desktop$ ./job1.hql
SLF4J: Class path contains multiple SLF4J bindings
SLF4J: Found binding in [jar:file:/Users/manuelgranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/Users/manuelgranchelli/Uni/BigData/hadoop-3.3.2/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/Users/manuelgranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/hive-common-2.3.9.jar!/hive-log4j2.properties Async: true
OK
Time taken: 5.216 seconds
OK
Time taken: 1.374 seconds
OK
Time taken: 0.038 seconds
OK
Time taken: 0.825 seconds
Loading data to table default.year_text
OK
Time taken: 1.49 seconds
Added resources: [/year_words.py]
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = manuelgranchelli_20220517114231_c25468d8-9851-426e-bc39-a23efd218398
```

Log Job 1 Hive - File (a)

2.3 Spark

L'implementazione Spark è stata effettuata tramite operazioni di pre-elaborazione e filtraggio per ottenere l'RDD contenente per ogni anno le dieci parole con il numero di occorrenze più elevato. Una volta caricato il file csv nell'RDD di input sono state eseguite, come nelle precedenti implementazioni, operazioni per pulire il testo e ottenere l'anno. Da questo RDD è stato creato un nuovo RDD contenente come chiave l'anno e la singola parola e come valore l'intero 1. L'intero 1 è stato utilizzato nel reduceByKey per poter contare per ogni anno le occorrenze di ogni parola. Successivamente sono state eseguite operazioni di ordinamento e raggruppamento per ottenere un RDD di output contenente per ogni anno le dieci parole più utilizzate con il relativo numero di occorrenze.

2.3.1 Pseudocodice

Algorithm Spark Job 1

```

1: input_RDD ← input file
2: time_text_RDD ← time, text
3: year_text_RDD ← converted time, cleaned text
4: year_words_RDD ← (year, word), 1
5: year_words_sum_RDD ← reduceByKey(year_words_RDD) and sort
6: output_RDD ← year, (word, count)
7: output_top10_RDD ← groupByKey(output_RDD) and get top 10 words

```

2.3.2 Script

Lo script utilizzato, *year_words.py*, è presente all'interno del **Repo GitHub**.

2.3.3 Output



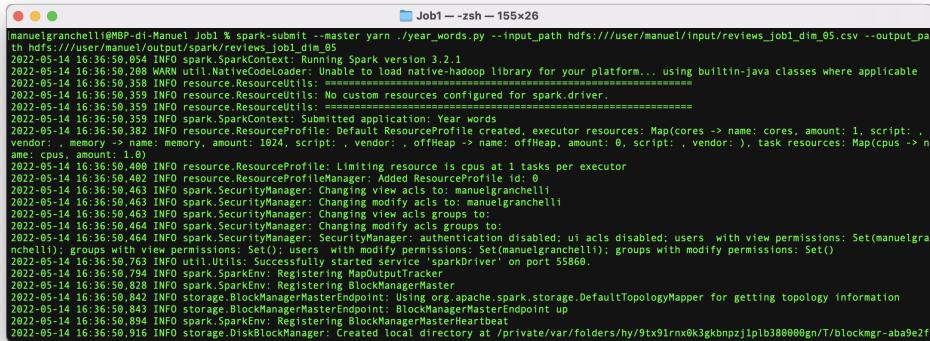
```

manuelgranchelli@MBP-d1-Manuel reviews_job1_dim_05 % head -50 part-00000 part-00001
==> part-00000 <==
('2012', [(('the', 336231), ('!', 282835), ('and', 230366), ('a', 215022), ('to', 182936), ('it', 172096), ('of', 144680), ('is', 132588), ('this', 125278),
('for', 98246)), ('2011', [(('the', 272642), ('!', 220675), ('and', 185338), ('a', 174164), ('to', 149285), ('it', 137435), ('of', 114326), ('is', 104891), ('this', 98666),
('for', 79303))], ('2010', [(('the', 140915), ('!', 13872), ('and', 96986), ('a', 90874), ('to', 75910), ('it', 68585), ('of', 60459), ('is', 55252), ('this', 50271), ('for',
42333))], ('2009', [(('the', 84717), ('!', 66534), ('and', 59357), ('a', 54991), ('to', 45935), ('it', 40861), ('of', 37294), ('is', 33974), ('this', 29453), ('for',
25858)), ('2008', [(('the', 53208), ('!', 38332), ('and', 36389), ('a', 34281), ('to', 27990), ('it', 23632), ('of', 23521), ('is', 20833), ('this', 17509), ('for',
15841))], ('2007', [(('the', 30354), ('!', 21846), ('and', 20864), ('a', 19587), ('to', 15898), ('it', 13487), ('of', 13275), ('is', 12534), ('this', 10483), ('in', 9
190))])
==> part-00001 <==
('2006', [(('the', 12501), ('and', 8281), ('!', 7767), ('a', 7656), ('to', 5981), ('of', 5561), ('is', 4922), ('it', 4921), ('in', 3637), ('this', 3498)), ('2005',
[('the', 2628), ('and', 1766), ('a', 1654), ('!', 1489), ('to', 1300), ('of', 1206), ('is', 1122), ('it', 1069), ('in', 892), ('this', 737))], ('2004',
[('the', 1536), ('and', 8681), ('!', 1383), ('to', 680), ('of', 687), ('is', 582), ('it', 521), ('in', 416))), ('2003',
[('the', 202), ('and', 189), ('!', 148), ('to', 126), ('of', 118), ('is', 105), ('it', 103), ('in', 88), ('this', 82))], ('2002',
[('the', 239), ('a', 164), ('and', 145), ('!', 126), ('is', 99), ('to', 73), ('it', 67), ('in', 64), ('this', 64)]), ('2001',
[('the', 30), ('to', 26), ('a', 20), ('and', 18), ('!', 17), ('it', 13), ('is', 9), ('in', 8), ('was', 8), ('of', 7))]), ('2000',
[('the', 66), ('of', 37), ('and', 30), ('to', 23), ('film', 21), ('that', 20), ('is', 19), ('a', 19), ('it', 12), ('an', 10)]), ('1999',
[('the', 6), ('this', 4), ('is', 4), ('to', 4), ('book', 3), ('i', 3), ('it', 3), ('son', 2), ('in', 2), ('he', 2)])

```

Ouput Job 1 Spark - File (a)

2.3.4 Log



```

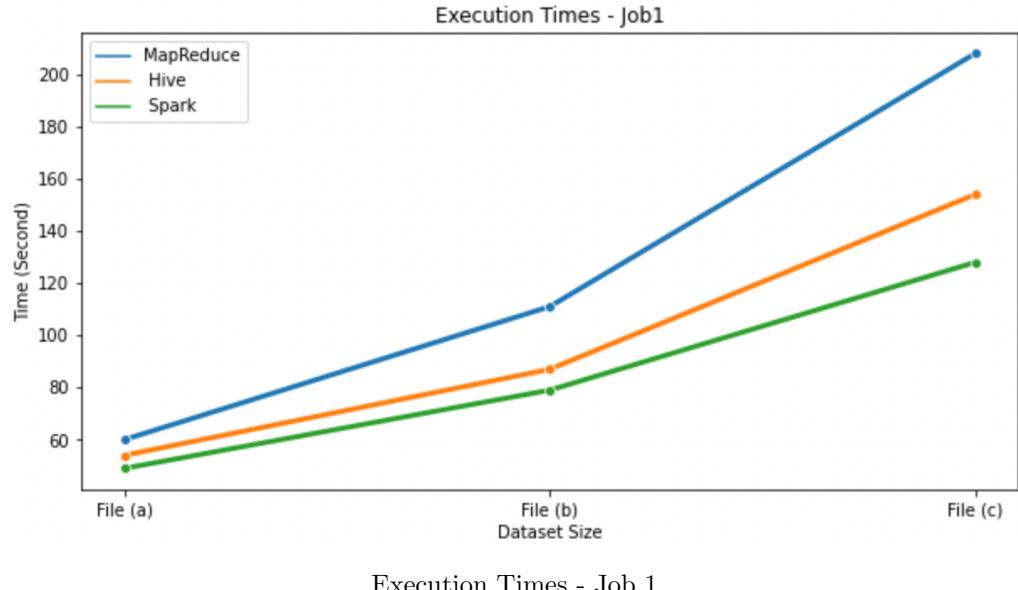
manuelgranchelli@MBP-d1-Manuel Job1 % spark-submit --master yarn ./year_words.py --input_path hdfs://user/manuel/input/reviews_job1_dim_05.csv --output_path
hdfs://user/manuel/output/spark/review_job1.dim.05
2022-05-14 16:36:50.054 INFO spark.SparkContext: Running Spark version 3.2.1
2022-05-14 16:36:50.208 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2022-05-14 16:36:50.359 INFO Resource.ResourceUtils: =====
2022-05-14 16:36:50.359 INFO Resource.ResourceUtils: ===== custom resources configured for spark.driver
2022-05-14 16:36:50.359 INFO Resource.ResourceUtils: =====
2022-05-14 16:36:50.359 INFO spark.SparkContext: Submitted application: Year words
2022-05-14 16:36:50.382 INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: . vendor: ., memory -> name: memory, amount: 1024, script: ., vendor: ., offheap -> name: offheap, amount: 0, script: ., vendor: .), task resources: Map(cpus -> n
ame: cpus, amount: 1, script: ., vendor: .)
2022-05-14 16:36:50.400 INFO resource.ResourceProfile: Limiting resource is cpus at 1 tasks per executor
2022-05-14 16:36:50.402 INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
2022-05-14 16:36:50.463 INFO spark.SecurityManager: Changing view acls to: manuelgranchelli
2022-05-14 16:36:50.463 INFO spark.SecurityManager: Changing modify acls to: manuelgranchelli
2022-05-14 16:36:50.463 INFO spark.SecurityManager: Changing view acls groups to:
2022-05-14 16:36:50.464 INFO spark.SecurityManager: Changing modify acls groups to:
2022-05-14 16:36:50.464 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(manuelgranchelli); groups with view permissions: Set()
2022-05-14 16:36:50.464 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with modify permissions: Set(manuelgranchelli); groups with modify permissions: Set()
2022-05-14 16:36:50.763 INFO util.Utils: Successfully started service 'sparkDriver' on port 55868.
2022-05-14 16:36:50.794 INFO spark.SparkEnv: Registering MapOutputTracker
2022-05-14 16:36:50.828 INFO spark.SparkEnv: Registering BlockManagerMaster
2022-05-14 16:36:50.842 INFO storage.BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
2022-05-14 16:36:50.843 INFO storage.BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
2022-05-14 16:36:50.894 INFO spark.SparkEnv: Registering BlockManagerMasterHeartbeat
2022-05-14 16:36:50.916 INFO storage.DiskBlockManager: Created local directory at /private/var/folders/hy/9tx91rnrx0k3gbnpzj1plb38000gn/T/blockmgr-aba9e2f

```

Log Job 1 Spark - File (a)

2.4 Tempi di esecuzione

Dal grafico si può osservare che l'implementazione Spark è la più veloce e MapReduce è la più lenta. I tempi di esecuzione crescono al crescere della dimensione del file e con il file (c) il tempo di esecuzione con MapReduce è quasi il doppio di quello con Spark.



3 Job 2

L'obiettivo del seguente Job è quello di restituire per ciascuno utente i prodotti preferiti (quelli con score più alto) fino ad un massimo di 5. Il risultato deve essere ordinato in base allo UserId. Nel seguente Job sono stati utilizzati i tre file di diversa dimensione generati precedentemente tutti aventi i seguenti tre campi: *ProductId*, *UserId* e *Score*.

3.1 MapReduce

L'implementazione MapReduce è stata svolta, come nel Job 1, creando due script, uno mapper ed un reducer. Il mapper legge il file csv una riga per volta ed esegue uno split della riga in tre parti. Ciascuna riga contiene il productId, lo userId e lo score e per ciascuna riga il mapper stampa una stringa contenente userId, productId e score.

La stringa viene letta dal reducer che crea un dizionario contenente per ciascuno utente, i prodotti recensiti con il relativo score. In seguito, per ciascun utente il reducer esegue un ordinamento dei prodotti in base allo score decrescente e stampa in output, ordinando in base allo userId, userId e la lista dei prodotti con i relativi score.

3.1.1 Pseudocodice

Algorithm Mapper Job 2

```
1: for line in stdin do
2:   line ← line.strip()
3:   productId, userId, score ← line.split()
4:   Output (userId, productId, score )
5: end for
```

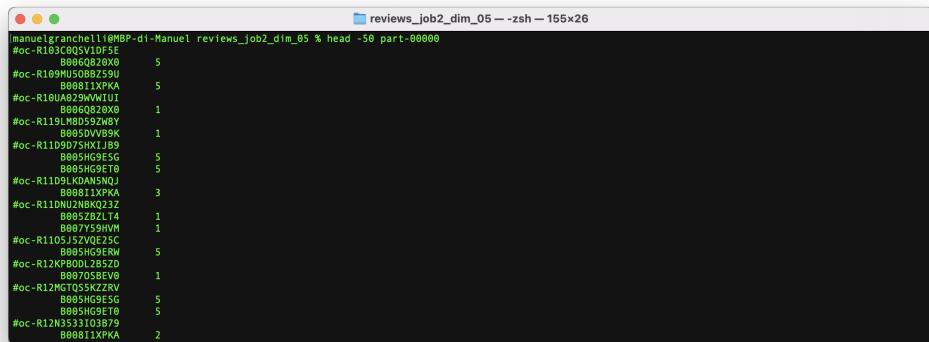
Algorithm Reducer Job 2

```
1: Initialize user_product_score dictionary
2: for line in stdin do
3:   userId, productId, score  $\leftarrow$  line.split()
4:   if userId not in user_product_score then
5:     Initialize user_product_score[userId] dictionary
6:   end if
7:   user_product_score [userId] [productId]  $\leftarrow$  score
8: end for
9: for userId in sort(user_product_score) do
10:   Output(userId)
11:   for productId in sort(user_product_score[userId]) do
12:     Output(productId, score)
13:   end for
14: end for
```

3.1.2 Script

I due script utilizzati sono *mapper.py* e *reducer.py* presenti all'interno del **Repo GitHub**.

3.1.3 Output



```
#oc-R103C805V1DFSE
  B0B6Q0B2X0 5
#oc-R169MUS0B8259U
  B0B6Q0B1AK 5
#oc-R169MUS0B8259U
  B0B6Q0B2X0 1
#oc-R19LMBD592WBY
  B0B6DVW89K 1
#oc-R1D9D7SHX1JB9
  B0B6Q0B1SG 5
  B0B6Q0B1SG 5
#oc-R1D9LKDANSQJ
  B0B811XPKA 3
#oc-R1DNU2NBKQ23Z
  B0B6Q0B2L1T4 1
  B0B7Y59HVM 1
#oc-R11052W2E95C
  B0B6Q0B1GVW 5
#oc-R12KPBDOL2B5ZD
  B0B670SBEV9 1
#oc-R12MGTQSSKZ2RV
  B0B6H9GE5G 5
  B0B6H9GE10 5
#oc-R12N533103B79
  B0B811XPKA 2
```

Ouput Job 2 MapReduce - File (a)

3.1.4 Log

```

manu@manu-OptiPlex-5090:~/Desktop$ hadoop jar $HADOOP_HOME/streaming/hadoop-streaming-3.3.0.jar -mapper mapper.py -reducer reducer.py -input /user/manu/input/reviews_05.csv -output /user/manu/output/reviews_job2.dim_05
2022-05-13 10:11:39.115 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2022-05-13 10:11:39.728 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2022-05-13 10:11:39.882 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2022-05-13 10:11:39.882 INFO impl.MetricsSystemImpl: JobMetrics2 metric system initialized
2022-05-13 10:11:39.874 WARN impl.MetricsSystemImpl: JobMetrics2 metric system may be initialized!
2022-05-13 10:11:40.428 INFO mapred.FileInputFormat: Total input files to process : 1
2022-05-13 10:11:40.588 INFO mapreduce.JobSubmitter: number of splits:1
2022-05-13 10:11:40.588 INFO mapreduce.JobSubmitter: Submitting application master job: job_local1689491742_0001
2022-05-13 10:11:40.778 INFO mapreduce.Job: Number of reduce tasks: 0 with tokens: []
2022-05-13 10:11:40.967 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2022-05-13 10:11:40.969 INFO mapreduce.Job: Running job: job_local1689491742_0001
2022-05-13 10:11:40.978 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2022-05-13 10:11:40.978 INFO mapred.LocalJobRunner: OutputCommitter set in arguments null
2022-05-13 10:11:40.979 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-05-13 10:11:40.979 INFO output.FileOutputCommitter: FileOutputCommitter cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-05-13 10:11:41.153 INFO mapred.LocalJobRunner: Waiting for map tasks
2022-05-13 10:11:41.153 INFO mapred.LocalJobRunner: Starting task: attempt_local1689491742_0001_m_000000_0
2022-05-13 10:11:41.183 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2022-05-13 10:11:41.183 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2022-05-13 10:11:41.195 INFO mapred.Task: Using ResourceCalculatorProcessTree currently is supported only on Linux.
2022-05-13 10:11:41.195 INFO mapred.Task: Using ResourceCalculatorProcessTree currently is supported only on Linux.
2022-05-13 10:11:41.314 INFO mapred.MapTask: numReduceTasks : 1
2022-05-13 10:11:41.357 INFO mapred.MapTask: (EQUATOR) 0 kv1 26214396(16485784)
2022-05-13 10:11:41.357 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2022-05-13 10:11:41.357 INFO mapred.MapTask: soft limit = 8388608
2022-05-13 10:11:41.357 INFO mapred.MapTask: total memory = 8388608
2022-05-13 10:11:41.357 INFO mapred.MapTask: kvstart = 26214396; length = 104857600
2022-05-13 10:11:41.357 INFO mapred.MapTask: kvstart = 26214396; length = 104857600

```

Log Job 2 MapReduce - File (a)

3.2 Hive

L'implementazione Hive è stata svolta utilizzando una tabella con tre campi *productId*, *userId* e *score* nella quale è stato caricato il file csv. Successivamente è stata creata la tabella di output contenente per ciascuna riga (*productId*, *userId*, *score*), ordinata in base allo *userId*. La creazione della tabella è stata fatta ordinando i prodotti per ciascun *userId* in base allo score e per ciascun utente è stato preso un numero massimo di prodotti pari a 5. Quindi, la tabella di output conterrà al massimo 5 righe per ciascun utente e ogni riga conterrà *userId*, *productId* e *score*.

3.2.1 Script

Lo script utilizzato è il seguente: *job2.hql* ed è presente all'interno del **Repo GitHub**.

3.2.2 Output

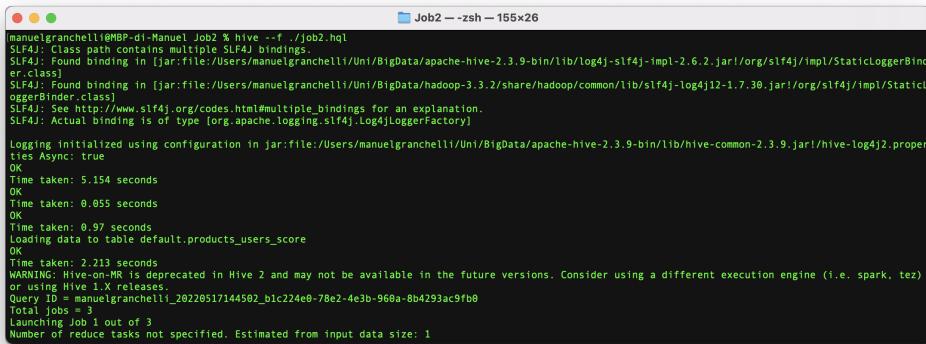
```

OK
#oc_R103C00SV1D15E B006QG02X0 5
#oc_R109MU0B06725U B00811XPXK 5
#oc_R109M03PVWV1UT B006QG02X0 1
#oc_R119LMB05P52W8Y B005DV89K 1
#oc_R11D9D7SHX1I89 B005HG9E5G 5
#oc_R11D9D7SHX1I89 B005HG9ET0 5
#oc_R11D9LKD0A5NQJ B00811XPXK 3
#oc_R11D0U2NBQK27 B005S2BLT4 1
#oc_R11D0U2NBQK27 B005S2BLT4 1
#oc_R1105JSZVQE25C B005HG9ERW 5
#oc_R12KPB0DL2B5ZD B00705BEV0 1
#oc_R12MGTQ55K2ZR B005HG9ET0 5
#oc_R12MGTQ55K2ZR B005HG9E5G 5
#oc_R12N5331J03B79 B00811XPXK 2
#oc_R12N5331J03B79 B00811XPXK 2
#oc_R13NNUL4EKLAFL B005HG9ERW 1
#oc_R13XYIJJGLTQC B005DV89K 4
#oc_R149FDXLARCWJ B00811XPXK 4
#oc_R14VLCY75K1B58 B006QG02X0 1
#oc_R14ZUK54VM00JS B006QG02X0 1
#oc_R152UR09M996EM B006QG02X0 2
#oc_R152UR09M996EM B006QG02X0 1
#oc_R155JB2SA58E17 B005HG9ET0 5
#oc_R150CG1KF5154F B006QG02X0 1
#oc_R163CP165RR150 B005HG9ERW 5

```

Output Job 2 Hive - File (a)

3.2.3 Log



```
manueigranchelli@MBP-di-Manuel: Job2 % hive --f ./job2.hql
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/manueigranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/Users/manueigranchelli/Uni/BigData/hadoop-3.3.2/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/Users/manueigranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/hive-common-2.3.9.jar!/hive-log4j2.properties Async: true
OK
Time taken: 5.154 seconds
OK
Time taken: 0.055 seconds
OK
Time taken: 0.97 seconds
Loading data to table default.products_users_score
OK
Time taken: 2.213 seconds
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez)
or, using Hive 1.X releases.
Query ID: manueigranchelli_20220517144502_b1c224e8-78e2-4e3b-968a-8b4293ac9fb0
Total jobs: 3
Launching Job 1 out of 3
Number of reduce tasks not specified. Estimated from input data size: 1
```

Log Job 2 Hive - File (a)

3.3 Spark

L'implementazione Spark è stata effettuata tramite operazioni di pre-elaborazione, filtraggio e raggruppamento per ottenere l'RDD contenente per ogni utente i prodotti recensiti con il più alto score. Una volta caricato il file csv nell'RDD di input è stata eseguita un'operazione di raggruppamento sulla chiave, quindi l'RDD avrà la seguente forma (**k**: userId, **v**: lista di prodotti). È stato creato un altro RDD nel quale viene eseguito l'ordinamento della lista dei prodotti in base allo score e nell'RDD di output il valore conterrà solo i primi 5 prodotti dalla lista con lo score più elevato. Infine, l'RDD di output è stato ordinato in base alla chiave, ovvero in base allo userId.

3.3.1 Pseudocodice

Algorithm Spark Job 2

```
1: input_RDD ← input file
2: user_product_RDD ← userId, productId, score
3: user_product_score_RDD ← userId, (productId, scoreId) and groupByKey
4: output_RDD ← userId, sort productId on score, score
5: output_top5_RDD ← userId, top 5 products, score
```

3.3.2 Script

Lo script utilizzato è il seguente: *user_products.py* ed è presente all'interno del **Repo GitHub**.

3.3.3 Output

```
manuelgranchelli@MBP-di-Manuel reviews_job2_dim_05 % head -50 part-00008
(#oc-R109MUS0BBZ5F5E, [(B006Q820K9, '5')))
(#oc-R109MUS0BBZ5F5E, [(B00811XPKA, '5'))])
(#oc-R110S5V0HJL4U1, [(B006Q820K9, '1'))])
(#oc-R119UW0D592W8Y, [(B005DVVB9K, '1'))])
(#oc-R11D9D7SHX1I89, [(B005HG9ESG, '5'), ('B005HG9ET0', '5'))])
(#oc-R11D9LK0AMN5QJ, [(B00811XPKA, '3'))])
(#oc-R11DN0U2NBQ2JZ, [(B007YS9HW, '1'), ('B005ZBZLT4', '1'))])
(#oc-R110S5ZV0E25C, [(B005HG9ERW, '5'))])
(#oc-R110S5ZV0E25C, [(B005HG9ERW, '1'))])
(#oc-R11NGC55KZ7RV, [(B005HG9ESG, '5'))])
(#oc-R12N531D03B79, [(B00811XPKA, '2'))])
(#oc-R13EBF129D8X88, [(B007YS9HW, '2'))])
(#oc-R13NNUL4EKLAFL, [(B005HG9ERW, '1'))])
(#oc-R13XY1YJ6GLT0C, [(B005DVVB9K, '4'))])
(#oc-R149FDXLARCWJ, [(B00811XPKA, '4'))])
(#oc-R150C165RR150, [(B006Q820K9, '1'))])
(#oc-R1AZUK5AVNOQJS, [(B006Q820K9, '1'))])
(#oc-R152U1R09M96EM, [(B006Q820K9, '2'))])
(#oc-R15343Zw0UTLNR, [(B005ZBZLSU, '1'))])
(#oc-R155JB2SA8E17, [(B005HG9ET0, '5'))])
(#oc-R150C165RR150, [(B006Q820K9, '1'))])
(#oc-R163CP165RR150, [(B005HG9ERW, '5'))])
(#oc-R1669TS0H07EP, [(B005ZBZLT4, '1'))]
```

Ouput Job 2 Spark - File (a)

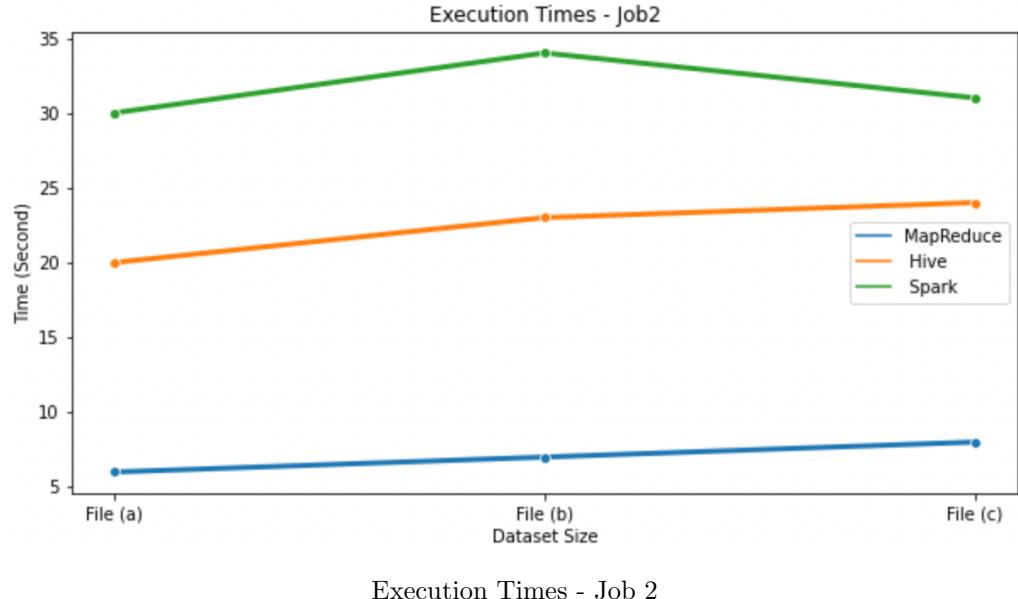
3.3.4 Log

```
manuelgranchelli@MBP-di-Manuel Job2 % spark-submit --master yarn ./user_products.py --input_path hdfs:///user/manuel/input/reviews_jobs_dim_05.csv --output _path hdfs:///user/manuel/output/spark/reviews_job2_dim_05
2022-05-14 18:35:14,449 INFO util.NativeCodeLoader: Loading Java Native Code Library: jvm股本库
2022-05-14 18:35:14,450 INFO util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
2022-05-14 18:35:14,451 INFO resource.ResourceUtils: =====
2022-05-14 18:35:14,775 INFO resource.ResourceUtils: No custom resources configured for spark.driver.
2022-05-14 18:35:14,775 INFO resource.ResourceUtils: =====
2022-05-14 18:35:14,893 INFO resource.SparkContext: Submitted applications: User products
2022-05-14 18:35:14,893 INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: ), task resources: Map(cpus -> n
ame: cpus, amount: 1), memory: memory, amount: 1024, script: , vendor: , offheap: amount: 0, script: , vendor: ), task resources: Map(cpus -> n
ame: cpus, amount: 1)
2022-05-14 18:35:14,822 INFO resource.ResourceProfile: Limiting resource is cpus at 1 tasks per executor
2022-05-14 18:35:14,824 INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
2022-05-14 18:35:14,898 INFO spark.SecurityManager: Changing view acls to: manuelgranchelli
2022-05-14 18:35:14,898 INFO spark.SecurityManager: Changing modify acls to: manuelgranchelli
2022-05-14 18:35:14,891 INFO spark.SecurityManager: Changing view acls groups to:
2022-05-14 18:35:14,891 INFO spark.SecurityManager: Changing modify acls groups to:
2022-05-14 18:35:14,891 INFO spark.SecurityManager: SecurityManager: authentication disabled; users with view permissions: Set(manuelgra
nchelli); groups with view permissions: Set(); users with modify permissions: Set(manuelgranchelli); groups with modify permissions: Set()
2022-05-14 18:35:15,186 INFO util.Utils: Successfully started service 'sparkDriver' on port 56508.
2022-05-14 18:35:15,215 INFO spark.SparkEnv: Registering MapOutputTracker
2022-05-14 18:35:15,215 INFO spark.SparkEnv: Registering BlockManagerMaster
2022-05-14 18:35:15,246 INFO storage.DiskBlockManager: Registering BlockManagerMasterEndpoint: org.apache.spark.storage.DefaultTopologyMapper for getting topology information
2022-05-14 18:35:15,267 INFO storage.DiskBlockManager: MasterEndpoint: BlockManagerMasterEndpoint up
2022-05-14 18:35:15,329 INFO spark.SparkEnv: Registering BlockManagerMasterHeartbeat
2022-05-14 18:35:15,343 INFO storage.DiskBlockManager: Created local directory at /private/var/folders/hy/9tx91rxn0k3gkbpzjplb38000gn/T/blockmgr-0e0b437
```

Log Job 2 Spark - File (a)

3.4 Tempi di esecuzione

Dal grafico si può osservare che l'implementazione MapReduce è la più veloce e Spark è la più lenta. Il tempo di esecuzione in tutte le implementazioni cresce di molto poco rispetto alla dimensione del file.



Execution Times - Job 2

4 Job 3

L'obiettivo del seguente e ultimo Job è quello di restituire coppie di utenti aventi gusti simili, ovvero che hanno recensito con un score superiore a 4 almeno tre prodotti in comune. L'output sarà costituito dalla coppia di utenti e dai prodotti in comune ordinato in base al primo utente. Anche in questo Job sono stati utilizzati i tre file di diversa dimensione generati precedentemente con i seguenti tre campi: *ProductId*, *UserId* e *Score*.

4.1 MapReduce

L'implementazione MapReduce è stata svolta, come nei precedenti casi, creando due script, uno script mapper ed uno reducer. Il mapper legge il file csv una riga per volta ed esegue, come nel Job 2, uno split della riga in tre parti. Ciascuna riga contiene il *productId*, lo *userId* e lo *score*. Per ciascuna riga stampa in output una stringa contenente *userId*, *productId* solo se lo *score* è ≥ 4 .

La stringa viene letta dal reducer che crea due dizionari, uno contenente per ciascuno utente, i prodotti recensiti e uno contenente per ciascun prodotto, gli utenti che hanno recensito quel prodotto. Successivamente dal secondo dizionario creato sono state generate coppie di utenti e per ogni coppia di utenti è stata effettuata l'intersezione dei prodotti recensiti. Per effettuare l'intersezione è stato utilizzato il primo dizionario creato. Se l'intersezione contiene più di tre prodotti, la coppia di utenti e l'intersezione dei prodotti vengono aggiunti, nel caso la coppia non fosse già presente, a un ulteriore dizionario che conterrà le coppie di utenti con i prodotti in comune. Quest'ultimo dizionario è stato ordinato e il reducer stampa in output le coppie di utenti e i prodotti in comune tra essi.

4.1.1 Pseudocodice

Algorithm Mapper Job 3

```
1: for line in stdin do
2:   line  $\leftarrow$  line.strip()
3:   productId, userId, score  $\leftarrow$  line.split()
4:   if score  $\geq 4$  then
5:     Output (userId, productId)
6:   end if
7: end for
```

Algorithm Reducer Job 3

```
1: Initialize product_user dictionary
2: Initialize user_product dictionary
3: Initialize user_similar_taste dictionary
4: for line in stdin do
5:   userId, productId, score  $\leftarrow$  line.split()
6:   if userId not in user_product then
7:     Initialize user_product[userId] set
8:   end if
9:   user_product [userId].add(productId)
10:  if productId not in product_user then
11:    Initialize product_user[productId] set
12:  end if
13:  product_user [productId].add(userId)
14: end for
15: for productId, userId in sort(product_user) do
16:   if len(userId) > 1 then
17:     couple_users  $\leftarrow$  create couple users from sort userId list
18:     for (user1, user2) in couple_users do
19:       intersection  $\leftarrow$  intersection products user1, user2
20:       if len(intersection)  $\geq 3$  and users not in dict then
21:         user_similar_taste [(user1, user2)]  $\leftarrow$  intersection
22:       end if
23:     end for
24:   end if
25: end for
26: for users in sort(user_similar_taste) do
27:   Output (users, productsId list)
28: end for
```

4.1.2 Script

I due script utilizzati sono *mapper.py* e *reducer.py* presenti all'interno del **Repo GitHub**.

4.1.3 Output

Ouput Job 3 MapReduce - File (a)

4.1.4 Log

Log Job 3 MapReduce - File (a)

4.2 Hive

L'implementazione Hive è stata effettuata in due differenti modi. In entrambi i casi è stata creata una tabella con tre campi *productId*, *userId* e *score* nella quale è stato caricato il file csv.

Inizialmente è stata svolta nel seguente modo: è stata creata una tabella nella quale è stato inserito il *JOIN* sul *productId* della tabella in input su se stessa. In seguito, è stata creata un'altra tabella, da quella precedente, contenente le coppie di utenti aventi numero di prodotti recensiti in comune maggiore di 2. Infine, è stata creata una tabella di output, ordinata in base al primo utente, contenente in ogni riga la coppia di utenti e il prodotto in comune. Quindi, nella tabella di output ci sarà un numero di righe contenente la stessa coppia di utenti pari al numero di prodotti che i due utenti hanno in comune. La seguente implementazione è stata testata sul file (a) (file dimezzato rispetto al dataset iniziale) e il Job ha impiegato più di 2 minuti per essere completato. Avendo ottenuto un tempo elevato rispetto alle implementazioni con MapReduce e Spark, è stata effettuata una nuova implementazione.

Nella seconda implementazione, invece, tramite un *SELECT TRANSFORM* e uno script scritto in Python è stata creata direttamente la tabella di output contenente in ogni riga

la coppia di utenti e una stringa contenente la lista di prodotti in comune tra la coppia di utenti. Lo script Python utilizzato è lo stesso del reducer utilizzato nell'implementazione MapReduce.

Confrontando i risultati si è osservato che con la seconda implementazione il processo è molto più veloce sul file (a) e per questo motivo si è scelto di utilizzare solamente la seconda implementazione anche per i successivi test. La differenza che si può osservare tra le due implementazioni è che nella seconda implementazione se c'è bisogno di ottenere i singoli productId sarà necessario eseguire un preprocessamento della stringa contenente i prodotti, cosa non necessaria nel caso di implementazione come nel primo caso.

4.2.1 Script

Lo script della prima implementazione è il seguente: *job3.hql* e quelli utilizzati nella seconda implementazione e con i quali sono stati effettuati i test sono i seguenti: *job3_script.hql* e *users_products.py*. Tutti presenti all'interno del **Repo GitHub**.

4.2.2 Output

```

OK
#oc-R19EJ3VEA88T60 A2YQ2Z16SF37N8 ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 A2RHVA2BTJ5VN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 AZMGNL420KHC ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 AM0TS5F11P ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 APDPAL11ZPYLN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 AS440EHT3KSPN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 A1U0QBFCER1P7VJ ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 A440YREFDM4IP ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 A32EOHILFZYXJP ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 A440YREFDM4IP ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-R19EJ3VEA88T60 #oc-RDMQLZAXFCUG ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A2YQ2Z16SF37N8 ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A3M6GTE1J4203T ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A1U0QBFCER1P7VJ ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A440YREFDM4IP ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG AM0TS5F11P ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG APDPAL11ZPYLN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A2RHVA2BTJ5VN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG AS440EHT3KSPN ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A32EOHILFZYXJP ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG AZMGNL420KHC ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A2U0QH9C9D9R ('B005HG9ET0', 'B005HG9ESG', 'B005HG9ERW')
#oc-RDMQLZAXFCUG A015561446ZNSD1JBESM A09XAZ1CLDTBD ('B005HG9ET0', 'B003PPFFIE', 'B002XG0S1')
#oc-RDMQLZAXFCUG A1YCNM7988EJSL ('B002R85LIUY', 'B002R85LI7YS', 'B002R8UANK')

```

Output Job 3 Hive - File (a)

4.2.3 Log

```

manuelgranchelli@0f-d1-Manuel: Job3 $ hive -f ./job3_script.hql
SLF4J: Class path contains multiple SLF4J bindings
SLF4J: Found binding in [jar:file:/Users/manuelgranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBind
er.class]
SLF4J: Found binding in [jar:file:/Users/manuelgranchelli/Uni/BigData/hadoop-3.3.2/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticL
oggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type (org.apache.logging.slf4j.Log4jLoggerFactory)

Logging initialized using configuration in jar:file:/Users/manuelgranchelli/Uni/BigData/apache-hive-2.3.9-bin/lib/hive-common-2.3.9.jar!/hive-log4j2.prop
erties Async: true
OK
Time taken: 5.059 seconds
OK
Time taken: 0.045 seconds
OK
Time taken: 0.787 seconds
Loading data to table default.products_users_score
OK
Time taken: 2.148 seconds
Added resources: [users_products.py]
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez)
or using Hive 1.X releases.
Query ID = manuelgranchelli_20220517162209_db2f7Ge1-b6c7-4823-9010-011e47350f3d
Total jobs = 1
Launching Job 1 out of 1

```

Log Job 3 Hive - File (a)

4.3 Spark

L'implementazione Spark è stata effettuata, come nelle precedenti implementazioni di Spark, partendo da un RDD di input nel quale viene caricato il file csv. L'RDD è stato filtrato per ottenere solo productId e userId con score ≥ 4 e dopo l'operazione di filtraggio l'RDD ha la seguente forma: (**k**: productId, **v**: userId). Questo RDD è stato raggruppato per chiave e la lista di utenti che si genera è stata ordinata. Da questa lista, in un nuovo RDD, sono state create le coppie di utenti.

Successivamente le coppie di utenti sono diventate la chiave per un nuovo RDD, quindi il seguente RDD ha la seguente forma: (**k**: couple user, **v**: productId). Con questo RDD è stato eseguito un raggruppamento per chiave così da ottenere per ogni coppia di utenti la lista dei prodotti recensiti in comune. In questa lista sono stati eliminati i duplicati e nell'RDD di output sono stati inseriti le coppie di utenti e la lista di prodotti in comune, ordinati in base al primo utente e avente lunghezza della lista dei prodotti in comune ≥ 3 .

4.4 Pseudocodice

Algorithm Spark Job 3

```

1: input_RDD  $\leftarrow$  input file
2: user_product_RDD  $\leftarrow$  productId, userId, filtered score  $\geq 4$ 
3: product_user_RDD  $\leftarrow$  productId, userId
4: product_user_RDD  $\leftarrow$  groupByKey() and sort userId list
5: product_couple_user_RDD  $\leftarrow$  productId, couple users list
6: users_product_RDD  $\leftarrow$  couple user, productId
7: users_product_RDD  $\leftarrow$  groupByKey()
8: output_RDD  $\leftarrow$  remove duplicate on productId list
9: output_cleaned_RDD  $\leftarrow$  filtered length productId list  $\geq 3$  and sortByKey()

```

4.4.1 Script

Lo script utilizzato è il seguente: *users_products.py* ed è presente all'interno del **Repo GitHub**.

4.4.2 Output

```

manuel.granchelli18MBP-d1-Manuel reviews_jobs3_dim_05 % head -50 part-00009
(''#oc-R19EJ3VEA88T60', '#oc-RODMLZAXFCUG'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A1UQBFCEIPVJ'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A2GMNLN42OKHC'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A2RNU42DJSVW'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A2RNU42DJSVZ'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A2YQZ165637N8'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A32EDHLFZYJEP'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A3M0ET1J4203T'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A440Y8EFD4M1P'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'AHUT5E5B8RDR'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'A440Y8EFD4M1P'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-R19EJ3VEA88T60', 'AS440EW73KSK'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A1UQBFCER1P7VJ'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A2MGNL420KHJC'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A2RNU42B7J5VN'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A2ZUNW9I2QKZNZ'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A32EDHLFZYJEP'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A32EDHLFZYJEP'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A3M0ET1J4203T'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'A440Y8EFD4M1P'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'AHUT5E98B0DR'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'APDA111ZPYLN'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#oc-RODMLZAXFCUG', 'AS44QEH73KSK'), ('B005H9ET0', 'B005HG9ERW', 'B005HG9ESG')
(''#AB555634RZNSLJ5EM', 'A2YKHAL2LD1AD'), ('B005PPFFIE', 'B009HTM6WK', 'B007K10BMS')
(''#A9533661HBB_HRFVRC', 'A1YCM798EJ5L'), ('B002R8J7YS', 'B002R8J4MK', 'B002R8J5UY')

```

Ouput Job 3 Spark - File (a)

4.4.3 Log

```

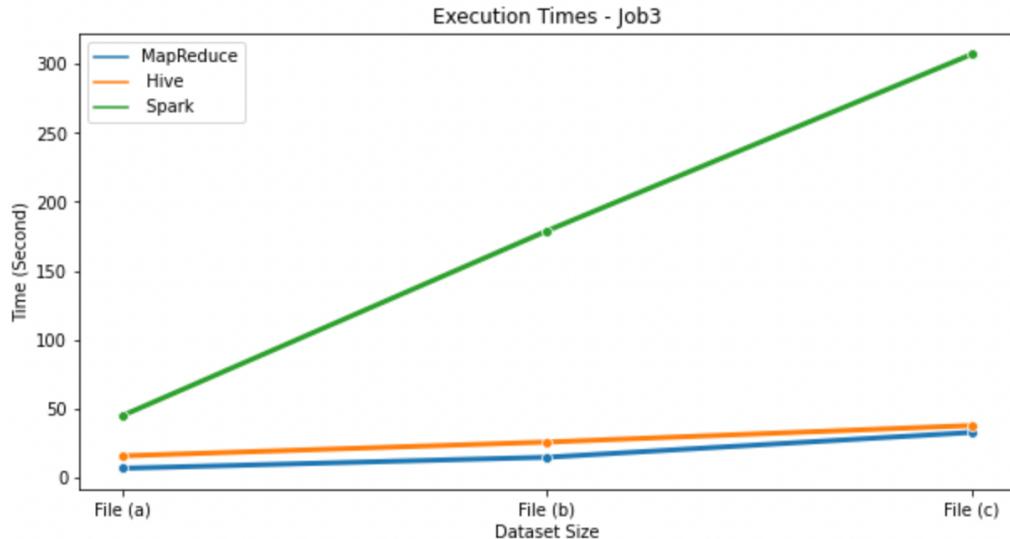
manuelegranchelli@MBP-di-Manuel:~/Manuel$ Job2 % spark-submit --master yarn ./user_products.py --input_path hdfs://user/manuel/input/reviews_dim_05.csv --output _path hdfs://user/manuel/output/spark/reviews_job2_dim_05
2022-05-14 18:35:14,499 INFO spark.SparkContext: Running Spark version 3.2.1
2022-05-14 18:35:14,500 INFO spark.SparkContext: Using JavaCPP Codec
2022-05-14 18:35:14,774 INFO resource.ResourceUtils: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2022-05-14 18:35:14,775 INFO resource.ResourceUtils: No custom resources configured for spark.driver
2022-05-14 18:35:14,775 INFO resource.ResourceUtils: =====
2022-05-14 18:35:14,775 INFO spark.SparkContext: Submitted application: User products
2022-05-14 18:35:14,893 INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: ), task resources: Map(cpus -> n
2022-05-14 18:35:14,893 INFO resource.ResourceProfile: Limiting resource is cpus at 1 tasks per executor
2022-05-14 18:35:14,824 INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
2022-05-14 18:35:14,898 INFO spark.SecurityManager: Changing view acls to: manuelegranchelli
2022-05-14 18:35:14,898 INFO spark.SecurityManager: Changing modify acls to: manuelegranchelli
2022-05-14 18:35:14,891 INFO spark.SecurityManager: Changing view acls groups to:
2022-05-14 18:35:14,891 INFO spark.SecurityManager: Changing modify acls groups to:
2022-05-14 18:35:14,891 INFO spark.SecurityManager: SecurityManager: authentication disabled; users with view permissions: Set(manuelegranchelli); groups with view permissions: Set(); users with modify permissions: Set(manuelegranchelli); groups with modify permissions: Set()
2022-05-14 18:35:15,215 INFO spark.SparkEnv: Successfully started service 'sparkDriver' on port 56598.
2022-05-14 18:35:15,252 INFO spark.SparkEnv: Registering MapOutputTracker
2022-05-14 18:35:15,267 INFO storage.BlockManagerMaster: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
2022-05-14 18:35:15,267 INFO storage.BlockManagerMaster: Local BlockManagerMasterEndpoint: LocalBlockManagerMasterEndpoint@10.0.0.1:56598
2022-05-14 18:35:15,320 INFO spark.SparkEnv: Registering BlockManagerMasterHeartbeat
2022-05-14 18:35:15,343 INFO storage.DiskBlockManager: Created local directory at /private/var/folders/hy/9tx91rx0k3gkbnpozjplb380000gn/T/blockmgr-0e0b437

```

Log Job 3 Spark - File (a)

4.5 Tempi di esecuzione

Dal Grafico si può osservare che l'implementazione Spark è la più lenta e il tempo di esecuzione cresce, al crescere della dimensione del file, molto di più rispetto alle altre due implementazioni. L'implementazione MapReduce è la più veloce e per quanto riguarda Hive nella sua prima implementazione era molto più lenta di Spark. Con la seconda implementazione, come si può osservare dal grafico, il tempo di esecuzione è di poco superiore a quello di MapReduce.



Execution Times - Job 3