Corso di laurea magistrale in
Ingegneria e Scienze Informatiche

Report on Big Data project: Codiv-19 papers analisys

Letizi Simone
Matricola: 887396

27th June 2020

Contents

1 Introduzione

Il seguente lavoro si propone di sviluppare 2 query su piattaforma Big Data con l'obiettivo finale di analizzare paper che trattano argomenti circa il COVID-19, al fine di comprendere quali sono le riviste con il maggior numero di pubblicazioni e quali sono gli autori più attivi su di esse. I paper sono suddivisi per categoria e la seconda query mira a identificare le parole più frequenti nelle diverse caregorie.

1.1 Descrizione del dataset

Il dataset è composto da più di 26.000 paper contenenti informazioni sui seguenti virus: Coronavirus, SARS, MERS e SARS-COV-2 Virus. Queste informazioni sono estratte dai dati pubblicati per la *Challenge Kaggle*. Su di essi è stato poi implementato un algoritmo di clustering per raggruppare i paper simili.
Il link per recuperare il dataset a cui si è fatto riferimento è il seguente: `https://www.kaggle.com/aestheteaman01/covid19-literary-analysis-dataset-covlad#COVID-Literature-Analysis.csv`. Il download include un singolo file csv di 783 MB.

1.1.1 Descrizione del file

Ogni record del file è caratterizzato dalle seguenti informazioni:

- paper_id: id univoco del paper;
- abstract: abstract del paper;
- body_text: testo completo del paper;
- authors: lista di autori separati da ',';
- title: titolo del paper;
- journal: rivista nella quale il paper è stato pubblicato;
- abstract_summary: riassunto dell'abstract;
- Labels: etichetta assegnata dall'algoritmo di clustering.

2 Preparazione dei dati

Il dataset utilizzato come input è memorizzato in HDFS al seguente link: `hdfs:/user/sletizi/examProject/dataset/COVID-Literature-Analysis_cleaned_partitioned.parquet`. Il file, originariamente in formato CSV, è stato convertito in formato parquet (specifico per Hadoop) con l'obiettivo di migliorare le performance.

2.1 Pre-processing

Rispetto al dataset scaricato dal link di cui sopra si sono eliminate, come suggerito dal creatore del dataset, le colonne 1 e 2, formate semplicemente da un numero progessivo utile all'indicizzazione.

3 Job #1: classifica degli autori più attivi nelle riviste più rilevanti

Il job risponde alla seguente interrogazione: "Dopo aver individuato le 100 riviste più importanti (sulla base del numero di pubblicazioni) stilare la classifica degli autori più attivi in tali riviste".

3.1 MapReduce

L'implementazione richiede che questo job venga sviluppato in 4 step map-reduce:

1. contare il numero di pubblicazioni per rivista;
2. ordinare per numero di pubblicazioni e limitare a 100 il risultato ottenuto;
3. recuperare i record contenenti le riviste rilevanti, splittare il campo authors per suddividere gli autori e contare il numero di pubblicazioni di ogni autore;
4. ordinare le pubblicazioni relative agli autori senza limitare il risultato finale.

Job #1: CountPublicationsPerJournalJob Il job prende in input il dataset memorizzato in HDFS in formato parquet; il dataset è suddiviso in 4 blocchi e di conseguenza vengono generati 4 task di map.

- **Mapper:** il ruolo del mapper è quello di mettere in output come chiave la rivista e 1 come valore, in maniera molto simile a quanto visto per il banale word count;
- **Reducer**: il reducer semplicemente somma i valori ottenuti per le stesse riviste.

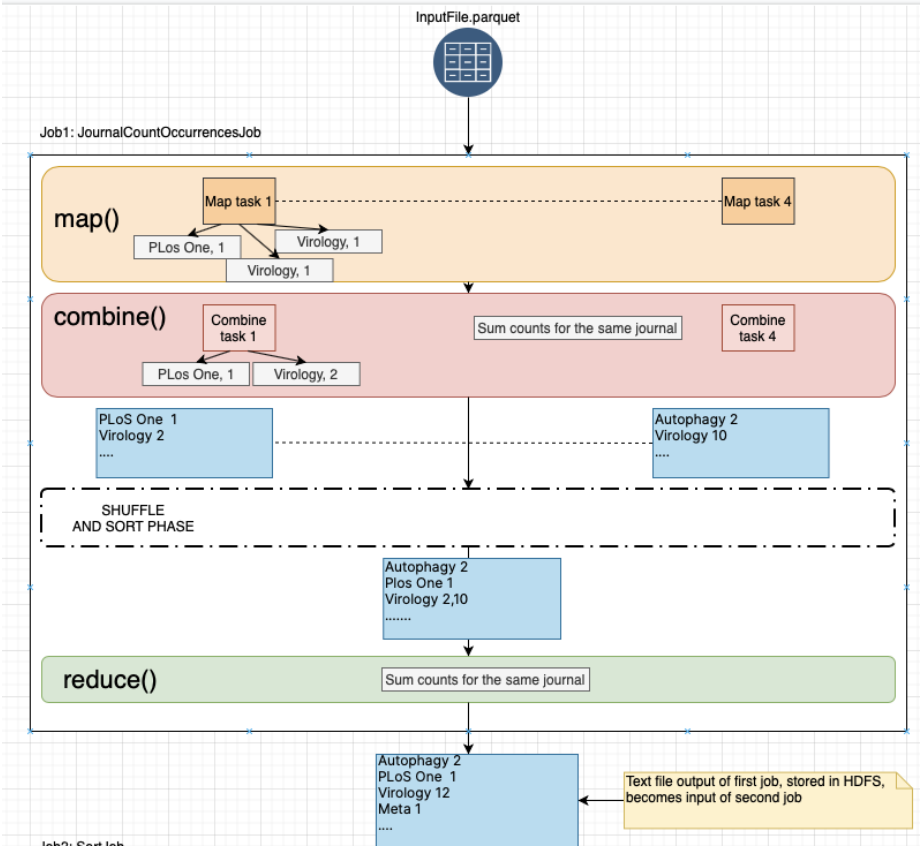


Figure 1: Job 1

Job #2: SortOccurrencesJob
Questo job si occupa di ordinare i valori ottenuti in input dal passo precedente.

- **Mapper:** il mapper inverte chiave e valore;
- **Reducer:** il reducer non riduce veramente i valori. Il vero lavoro viene fatto nella fase di shuffle and sort, a seguito dell'inversione di chiave e valore effettuata dal mapper. MapReduce infatti ordina automaticamente per chiave tra il passo di map e di reduce. Il fatto di utilizzare un solo reducer fa si che i risultati arrivino all'unico reducer totalmente ordinati, e non parzialmente.

Di default MapReduce ordina le chiavi in ordine crescente. Il framework, tuttavia, permette di specificare un Comparator differente a cui demandare il criterio di ordinamento delle chiavi.

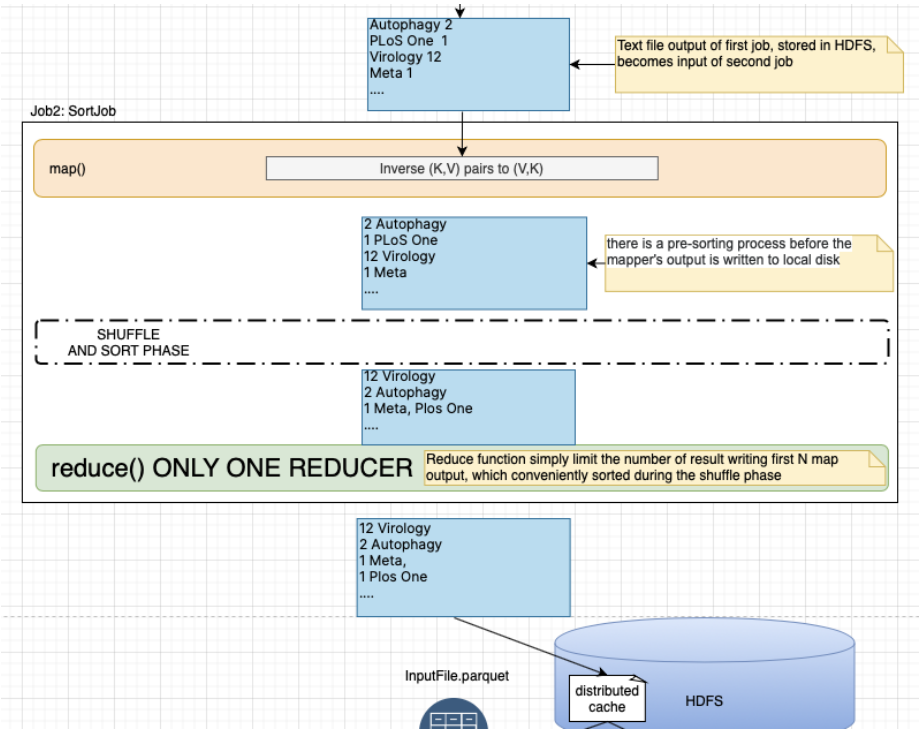


Figure 2: Job 2

Job #3: CountPublicationsPerAuthorReplicatedJoinJob
Questo job ha la scopo di scartare i record che hanno un valore per il campo rivista che non compare nella top 100.
Considerando il risultato del precedente job come nuovo dataset sarebbe necessaria un'operazione di join. L'uso del replicated-join, invece del classico join reduce-side, elimina la necessità di trasmettere in rete quei record il cui valore del campo journal non risulterebbe nella top 100. Questi verrebbero poi scartati nel passo di reduce. Il dataset contenente le top 100 riviste è piccolo abbastanza da poter essere memorizzato nella memoria dei task di map. Questo viene infatti pushato nella cache distribuita e replicato sui nodi del cluster. Il Join viene svolto interamente nella fase di map, utilizzando il dataset più grande (il dataset completo utilizzato nel job 1) come input del job. L'uso della Distributed Cache è un meccanismo utilizzato da Hadoop per distribuire files ai worker node in cui vengono istanziati i task di map e di reduce.

- **Mapper:** il mapper è responsabile di leggere il file dalla cache durante la fase di setup e memorizzarlo in memoria come tabella di look-up. Di ogni record processato viene verificata la presenza del valore del campo "journal" in memoria. Qualora vi fosse, il campo autore viene splittato e per ogni autore viene emessa una coppia chiave valore con il nome dell'autore come chiave e 1 come valore;

- **Reducer:** il reducer somma i valori per uno stesso autore ed emette una coppia (a,t) dove a è il nome dell'autore e t indica le occorrenze totali di a.

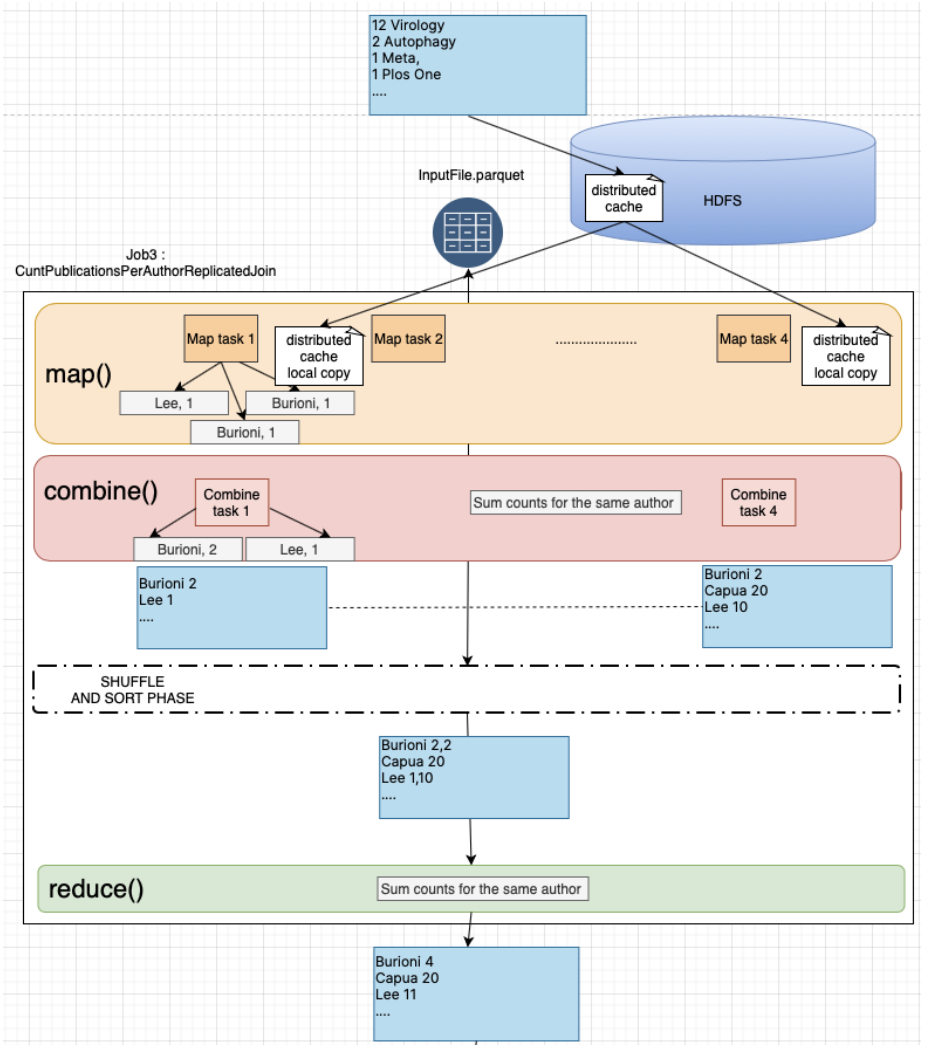


Figure 3

job #4: SortOccurrencesJob
Il job di sort già utilizzato nel Job #2 è stato parametrizzato e riutilizzato in questo job; l'unica differenza è che in questo caso il risultato non richiedeva di

essere limitato.

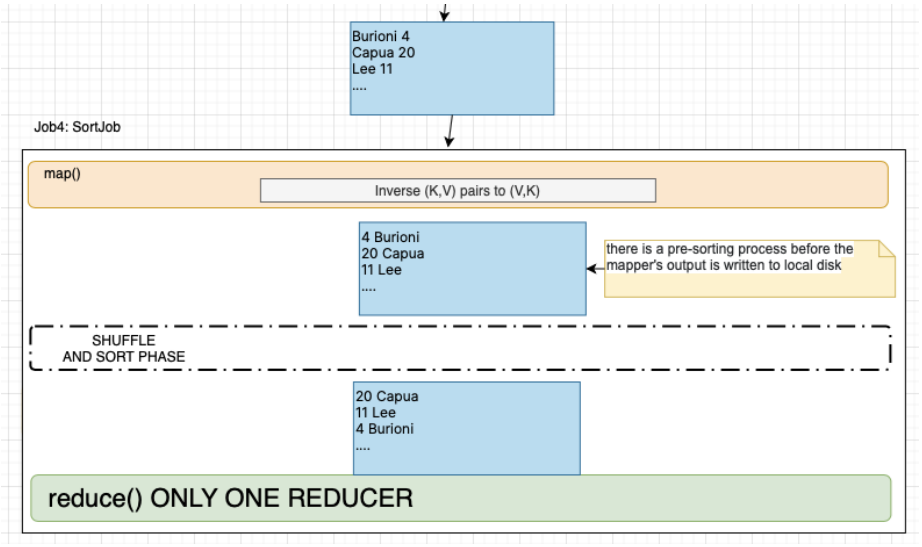


Figure 4

3.1.1 Valutazione delle performance

- JOIN: come citato sopra, il replicated join permette sicuramente di evitare lo shuffling inutile di dati, riducendo sensibilmente il lavoro del reducer;
- COMBINER: la funzione di aggregazione utilizzata nel reducer è associativa e commutativa; l'uso del combiner permette quindi di effettuare una fase di pre-aggregazione del risultato in fase di map. Questo riduce il traffico in rete e il numero di dati intermedi, con un conseguente minor numero di valori da gestire da parte del reducer. Come si può vedere nella tabella 1 nel primo Job i risultati pre-aggregati sono decisamente maggiori; nel terzo job si riduce solamente del 20% circa;
- PARTITIONER: una possibile miglioria che si sarebbe potuta implementare è quella di creare un partitioner personalizzato in quanto l'HashPartitioner utilizzato di default non tiene in considerazione il nu-

	Reduce input Records
Job1WithCombiner	5079
Job1WithoutCombiner	26044
Job3WithCombiner	32901
Job3WithoutCombiner	41092

Table 1: Effetto dell'uso del combiner

mero di valori per chiave. Come si può vedere dagli output ottenuti sia gli autori che le riviste (usati come chiavi) sono decisamente sbilanciati.

3.1.2 Tempo di esecuzione

La tabella mostrata in figura 8 mette a confronto diverse esecuzioni che differiscono per numero di task di reduce allocati e per l'uso del combiner. Si può notare come l'uso del combiner riduca il tempo di shuffle medio quasi in tutte le esecuzioni. Per quanto riguarda il numero di task di reduce, il sistema ne allocava di default 20, ma questo valore risulta nettamente sovradimensionato. Oltre ad essere inutile avere tanti task di reduce che scrivono in output su HDFS dei file di dimensioni notevolmente ridotte, vi è un grosso overhead per la creazione dei task.

JobName	ApplicationID	real Elapsed (s)	AvgMap	AvgShuffle	AvgReduce	#MapTask#ReduceTask	Combiner
CountPublicationsPerJournalJob	application_1583679666662_2979	23	11	4	0	4M-1R	si
	application_1583679666662_3011	25	11	5	0	4M-1R	no
	application_1583679666662_3015	26	12	6	0	4M-2R	si
	application_1583679666662_3023	25	10	6	1	4M-2R	no
	application_1583679666662_3113	41	11	6	0	4M-20R	si
SortOccurrencesJob	application_1583679666662_2980	16	5	5	0	1M-1R	/
	application_1583679666662_3012	17	6	4	0	1M-1R	/
	application_1583679666662_3016	18	7	5	0	2M-1R	/
	application_1583679666662_3024	18	7	5	1	2M-1R	/
	application_1583679666662_3114	30	7	5	0	20M-1R	/
ublicationsPerAuthorReplicatedJ	application_1583679666662_2981	23	10	5	1	4M-1R	si
	application_1583679666662_3013	27	12	6	1	4M-1R	no
	application_1583679666662_3017	27	12	5	1	4M-2R	si
	application_1583679666662_3025	28	13	6	1	2M-1R	no
	application_1583679666662_3115	44	10	8	1	4M-20R	si
SortOccurrencesJob	application_1583679666662_2982	17	6	4	0	1M-1R	/
	application_1583679666662_3014	19	6	5	0	1M-1R	/
	application_1583679666662_3018	20	8	5	0	2M-1R	/
	application_1583679666662_3026	20	7	5	0	2M-1R	/
	application_1583679666662_3116	30	7	5	0	20M-1R	/

ApplicationIDs		total elapsed (s)
application_1583679666662_2979/80/81/82	1-1-true	79
application_1583679666662_3011/12/13/14	1-1-false	88
application_1583679666662_3015/16/17/18	2-2-true	91
application_1583679666662_3023/24/25/26	2-2-false	91
application_1583679666662_3113/14/1516	20-20-true	145

Figure 5: Tempi di esecuzione con differente numero di task di reduce e uso di combiner

3.1.3 Output

Viene mostrato l'output delle prime 40 righe ottenuto digitando i comandi:

```
hadoop fs -cat /user/sletizi/examProject/mapreduce/output/
CountingSortedOutput.txt/part-r-00000|head -n 40
```

```
hadoop fs -cat /user/sletizi/examProject/mapreduce/output/
AuthorsPubsSortedOutput.txt/part-r-00000|head -n 40
```

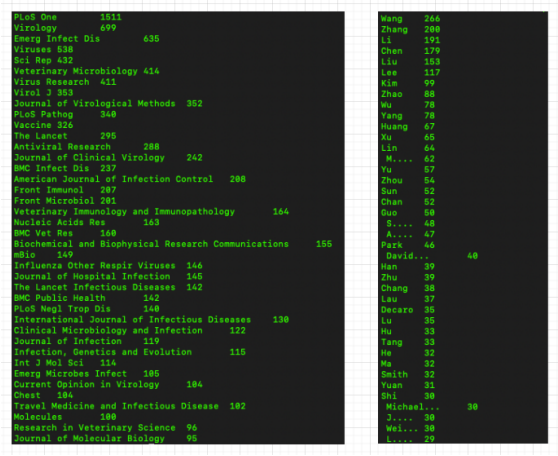


Figure 6: Output files:
CountingSortedOutput.txt - AuthorsPubsSortedOutput.txt

3.2 Spark

Il job implementato in Spark si divide nei seguenti passi:

- **caricare e ripulire il dataset**: caricare il dataset in formato parquet, eliminare le colonne che non sono necessarie per svolgere l'analisi in modo da migliorare le performance, e convertire l'RDD utilizzando una classe PublicationData che facilita le fasi successive;

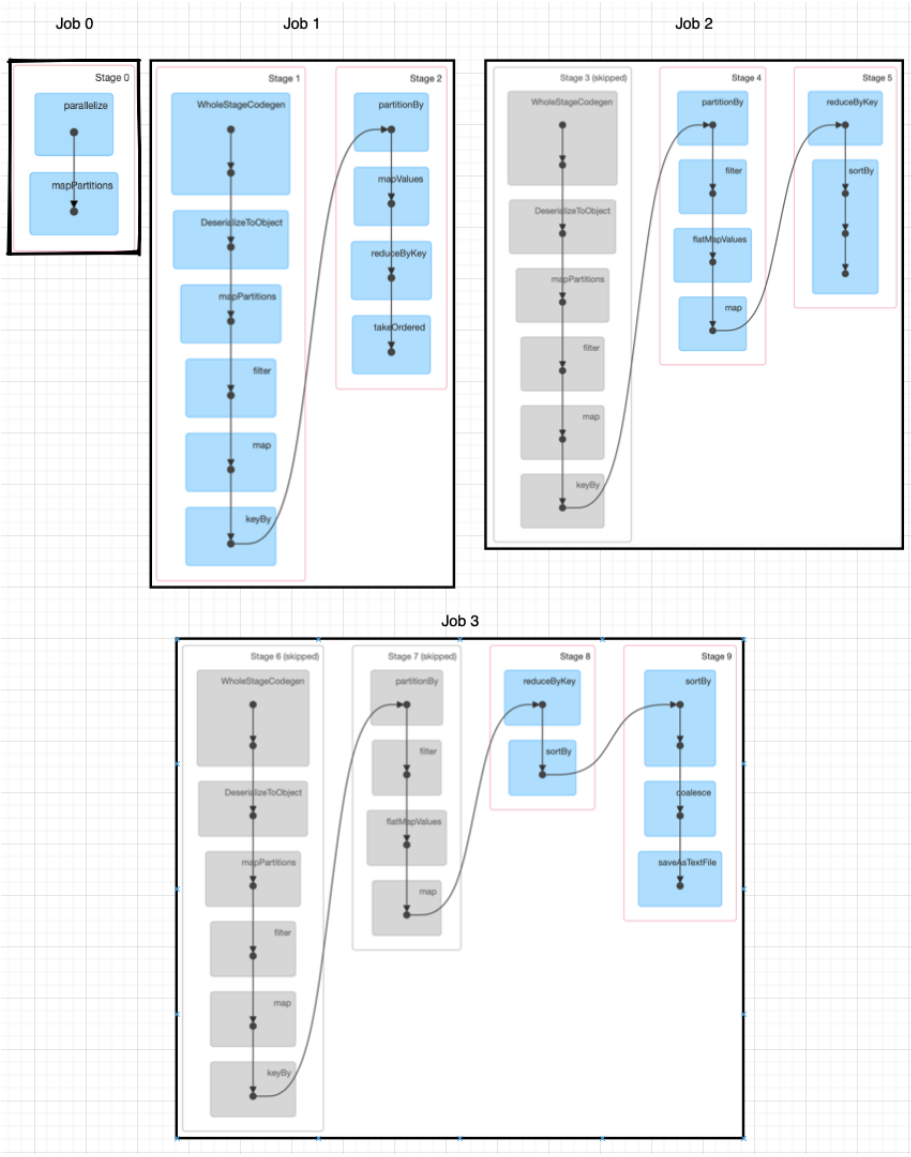
- **contare il numero di pubblicazioni per ogni giornale**: nel secondo job l'RDD viene trasformato in RDD chiave-valore attraverso l'operazione keyBy ponendo come chiave la rivista (campo journal).
  Successivamente, attraverso l'operazione `partitionBy` viene applicato un criterio di partizionamento (utilizzando HashPartitioner e settando il numero di partizioni a 12) in modo da muovere i record con gli stessi valori del campo journal nelle stesse partizioni e quindi facilitare le operazioni successive eliminando lo shuffling di dati. Attraverso l'operazione `mapValues` il valore viene posto a 1 senza perdere il criterio di partizionamento precedentemente impostato. Sull'RDD ottenuto viene prima chiamata l'operazione `reduceByKey` per aggregare i valori e poi richiamata l'azione `takeOrdered` per recuperare i primi 100 elementi specificando il criterio

di ordinamento. La mappa ottenuta viene inviata in broadcast ad ogni executor;

- **contare il numero di pubblicazioni per ogni autore:** a partire dall'RDD chiave-valore del job precedente vengono filtrati i record la cui chiave (journal) è contenuta nella broadcast variable. Attraverso l'operazione di `flatMapValues` vengono create più coppie chiave/valore per ogni autore presente nel campo authors. Si otterrà quindi un RDD con delle coppie (journal, author). Attraverso l'operazione di `map` a questo RDD viene impostata come chiave quello che prima era il valore (author) e come valore 1; cambiando il valore della chiave l'operazione di map provoca lo shuffle dei dati e un cambiamento di stage. I valori appartenenti alla stessa chiave vengono aggregati attraverso l'operazione di `reduceByKey` e poi ordinati in ordine decrescente attraverso l'operazione di `sortBy`;
- **salvare il risultato:** l'RDD ottenuto dai passi precedenti viene memorizzato in HDFS.

Il DAG viene mostrato in figura 7.

Figure 7: DAG


Job 0
Job 1
Job 2
Stage 0
parallelize
mapPartitions
Stage 1
WholeStageCodegen
DeserializeToObject
mapPartitions
filter
map
keyBy
Stage 2
partitionBy
mapValues
reduceByKey
takeOrdered
Stage 3 (skipped)
WholeStageCodegen
DeserializeToObject
mapPartitions
filter
map
keyBy
Stage 4
partitionBy
filter
flatMapValues
map
Stage 5
reduceByKey
sortBy
Job 3
Stage 6 (skipped)
WholeStageCodegen
DeserializeToObject
mapPartitions
filter
map
keyBy
Stage 7 (skipped)
partitionBy
filter
flatMapValues
map
Stage 8
reduceByKey
sortBy
Stage 9
sortBy
coalesce
saveAsTextFile

3.2.1 Tempo di esecuzione e ottimizzazioni

Il job è stato reso parametrizzabile in modo da valutare diversi tipi di esecuzione. I parametri in ingresso riguardano:

- il tipo di join:
  - 1 per utilizzare l'operazione `join` di Spark;
  - 2 per utilizzare il meccanismo delle broadcast variables sopra descritto;
  - 3 per utilizzare il meccanismo delle broadcast variables ma utilizzando l'operatore `map` invece di `keyBy` per costruire un RDD chiave-valore;
- l'utilizzo del criterio di partizionamento:
  - 0 per non utilizzare il criterio di partizionamento;
  - N per utilizzare il criterio di partizionamento Hash con N partizioni;
- l'utilizzo del meccanismo di caching dell'RDD:
  - true per mettere in cache l'RDD partizionato;
  - false per non utilizzare il meccanismo di caching;

Analizzando la tabella 2 si può notare come tra i diversi approcci vi sia una differenza impercettibile, sia per quanto riguarda l'uso del caching che del partizionamento. Ciò che appare con un po' più di evidenza è lo svantaggio di utilizzare un criterio di partizionamento con più partizioni del previsto in presenza di un solo core per executor. L'executor infatti, si ritroverà sequenzialmente a dover eseguire dei task che sono piccolissimi (vista la limitata grandezza dei dati per ciascun task).

AppIDs	parameters	executor cores	execution time
application_1583679666662_3331	1 - 0 - true	1	51s
application_1583679666662_3330	1 - 0 - true	2	48s
application_1583679666662_3335	1 - 0 - false	1	50s
application_1583679666662_3334	1 - 0 - false	2	49s
application_1583679666662_3336	1 - 12 - true	1	53s
application_1583679666662_3337	1 - 12 - false	1	52s
application_1583679666662_3338	1 - 12 - true	2	52s
application_1583679666662_3339	1 - 12 - true	3	54s
application_1583679666662_3340	1 - 12 - false	2	52s
application_1583679666662_3341	1 - 12 - false	3	49s
application_1583679666662_3343	2 - 0 - false	3	56s
application_1583679666662_3344	2 - 0 - false	2	50s
application_1583679666662_3345	2 - 0 - false	1	50s
application_1583679666662_3346	2 - 0 - true	2	50s
application_1583679666662_3347	2 - 12 - true	1	52s
application_1583679666662_3348	2 - 12 - true	2	49s
application_1583679666662_3349	2 - 12 - true	3	48s
application_1583679666662_3350	2 - 12 - false	3	48s
application_1583679666662_3351	3	3	49s

Table 2: Differenti tempi di esecuzione del job Spark con parametri differenti

3.2.2 Output

Viene mostrato l'output delle prime 40 righe ottenuto digitando i comandi:

```
   hadoop fs -cat /user/sletizi/examProject/spark/output/
authors_rank.txt/part-r-00000|head -n 40

   hadoop fs -cat /user/sletizi/examProject/spark/output/
journals_rank.txt/part-r-00000|head -n 40
```

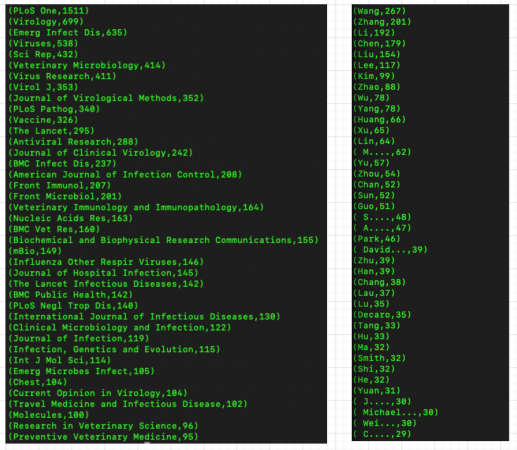


Figure 8: Output files:
journals_rank.txt - authors_rank.txt

3.3 MapReduce, Spark, SparkSQL

Il job è stato implementato in MapReduce, Spark e SparkSQL in modo da avere una visione completa dei differenti mezzi a disposizione, pur non avendo dedicando particolarmente attenzione alle ottimizzazioni in SparkSQL. Sicuramente MapReduce si è rivelato lo strumento più macchinoso da implementare e meno performante dei 3. Spark dalla sua, oltre ad offrire un maggior livello di astrazione e migliori performance rispetto a MapReduce, permette di avere un'idea più approfondita di ciò che avviene operativamente a differenza di SparkSQL. Di seguito si riportano i link alla history di YARN:

- MapReduce
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1583679666662_2979`
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1583679666662_2980`
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1583679666662_2981`
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1583679666662_2982`

- Spark
  - `http://isi-vclust0.csr.unibo.it:8088/cluster/app/application_1583679666662_3351`
- SparkSQL
  - `http://isi-vclust0.csr.unibo.it:8088/cluster/app/application_1583679666662_3431`

	Elapsed	**App ID**
MapReduce	1.19m	application_1583679666662_2979/80/81/82
Spark	40s	application_1583679666662_3351
SparkSQL	54s	application_1583679666662_3431

4 Job #2: Frequenza media delle parole nei paper di una stessa categoria (facoltativo)

Il job corrisponde alla seguente interrogazione: "calcolare, per ogni categoria (campo Labels), le 10 parole con la frequenza media più elevata (tra i paper di una stessa label) all'interno dei riassunti degli abstract (campo abstract_summary)".
Considerando la Term-Frequency (TF) all'interno di un paper come:
`numero di occorrenze del termine / numero dei termini totali del paper`
è necessario calcolare:

- la TF per ogni parola in ogni paper; per ottenerla ho bisogno di calcolare:
  - le occorrenze di ogni termine presente nel paper;
  - il numero di termini per ogni paper (al netto delle stop word);
- la TF media di una parola per i paper di una stessa categoria; per ottenerla ho bisogno di calcolare:
  - la somma delle TF di una stessa parola per i paper di una stessa categoria;
  - il numero dei paper per ogni categoria.

Esempio:

paper_id	abstract_summary	Labels
id_1	Il Covid è una malattia infettiva che colpisce l'apparato respiratorio. I test per il COVID sono iniziati in tutte le città del mondo	1
id_2	La maggior parte delle persone che contraggono il virus sviluppa sintomi lievi o moderati e guarisce senza aver bisogno di particolari cure.	1
id_3	Si può contrarre l'infezione respirando il virus se ci si trova nelle immediate vicinanze di una persona affetta da COVID	1
id_4	Le zanzare non trasmettono il COVID, lo conferma uno studio.	2
id_5	....	2
_6	....	2

Table 3: Esempio del calcolo della TF media per categoria

Volendo calcolare la TF media della parola covid all'interno della categoria 1 (per semplicità al momento trattando le stop word come le altre parole; nella query queste non vengono considerate nel calcolo della lunghezza del paper). Calcoliamo prima la TF di covid per ogni paper:

$$TF(covid, ID\_1) = 2/23 \tag{1}$$

$$TF(covid, ID\_1) = 0/22 \tag{2}$$

$$TF(covid, ID\_1) = 1/20 \tag{3}$$

$$Mean - TF(covid, Label1) = (2/23 + 1/20 + 0/22)/3 \tag{4}$$

Inoltre, si è deciso di aggiungere una query che calcolasse le 10 parole con il numero di occorrenze medie maggiori per tutti i paper etichettati con una stessa Label (senza tenere in considerazione quindi la lunghezza del paper come nel caso della TF), sfruttando alcuni RDD/DF già creati per calcolare la TF.

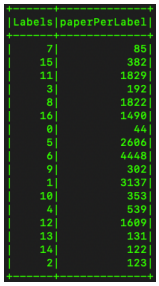
4.1 SparkSQL

Il job, inizialmente realizzato in SparkSQL, si compone dei seguenti passi:

1. **caricamento di un DF per le stop word** a partire da un file di testo memorizzato in HDFS;

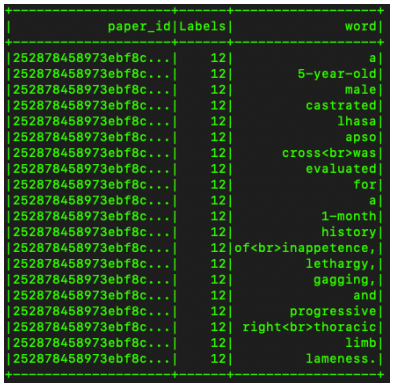
2. **caricamento del file parquet relativo al dataset in un DF**, rimuovendo le colonne che non coinvolgono la query in questione e scartando record con eventuali valori null nei campi desiderati;

3. **calcolo dei paper per ogni label** (`groupBy` label e `count` come funzione di aggregazione);



Labels	paperPerLabel
7	85
15	382
11	1829
3	192
8	1822
16	1490
0	44
5	2606
6	4448
9	302
1	3137
10	353
4	539
12	1609
13	131
14	122
2	123

4. **distinzione delle parole di ogni paper** splittando il campo abstract_summary in modo da ottenere una riga per ogni termine presente nel paper (funzione `extend` di SparkSQL) e trasformando le parole in lowerCase (funzione `lower` di SparkSQL);



paper_id	Labels	word
252878458973ebf8c...	12	a
252878458973ebf8c...	12	5-year-old
252878458973ebf8c...	12	male
252878458973ebf8c...	12	castrated
252878458973ebf8c...	12	lhasa
252878458973ebf8c...	12	apso
252878458973ebf8c...	12	cross was
252878458973ebf8c...	12	evaluated
252878458973ebf8c...	12	for
252878458973ebf8c...	12	a
252878458973ebf8c...	12	1-month
252878458973ebf8c...	12	history
252878458973ebf8c...	12	of inappetence,
252878458973ebf8c...	12	lethargy,
252878458973ebf8c...	12	gagging,
252878458973ebf8c...	12	and
252878458973ebf8c...	12	progressive
252878458973ebf8c...	12	right thoracic
252878458973ebf8c...	12	limb
252878458973ebf8c...	12	lameness.

5. **eliminazione delle stop word** effettuando un left-anti-join con il DF delle stopWords;

paper_id	Labels	word
79c0f8dbc31024ca3...	13	ebola
79c0f8dbc31024ca3...	13	virus
79c0f8dbc31024ca3...	13	(ebov)
79c0f8dbc31024ca3...	13	entry
79c0f8dbc31024ca3...	13	requires
79c0f8dbc31024ca3...	13	virion surface...
79c0f8dbc31024ca3...	13	glycoprotein
79c0f8dbc31024ca3...	13	(gp)
79c0f8dbc31024ca3...	13	composed
79c0f8dbc31024ca3...	13	trimer of
79c0f8dbc31024ca3...	13	heterodimers
79c0f8dbc31024ca3...	13	(gp1/gp2).
79c0f8dbc31024ca3...	13	gp1
79c0f8dbc31024ca3...	13	subunit contains
79c0f8dbc31024ca3...	13	heavily
79c0f8dbc31024ca3...	13	glycosylated
79c0f8dbc31024ca3...	13	domains,
79c0f8dbc31024ca3...	13	glycan cap
79c0f8dbc31024ca3...	13	mucin-like
79c0f8dbc31024ca3...	13	domain

6. **calcolo della lunghezza di ogni paper** a partire dal DF, al netto delle stop word, utilizzando `groupBy` paper_id e `count` come funzione di aggregazione;

paper_id	paperLength
27bafe2069fadba96...	60
f00106cad50635bb1...	55
6da13d90743ba52bd...	61
4b2ef217cb3fedcca...	59
11f3022ad1def6dfb...	55
525c07cc621b7185f...	56
0be7b06a39aa31b27...	50
7bfb902f2434b4667...	58
319da4b3a356bea37...	56
0266337807a2aab25...	61
1a7d600387654c031...	66
417006f8744a4d806...	55
136f5dd69cd65fb08...	56
e8f4d22d624e8deec...	57
024d0bef5417a110f...	57
34081c03e48678c7d...	58
5f36e6c3da64e8d95...	51
fffaed7e9353b7df6...	61
e1e50e72368b17371...	29
dc502e4494d5ef1e5...	50

7. **calcolo delle occorrenze totali di ogni termine per ogni paper** a partire dal DF al netto delle stop word, inserendo nella clausola di `groupBy` paper_id, Labels, word e usando `count` come funzione di aggregazione;

paper_id	word	Labels	Occurrences
79c0f8dbc31024ca3...	ebola	13	1
79c0f8dbc31024ca3...	virus	13	1
79c0f8dbc31024ca3...	(ebov)	13	1
79c0f8dbc31024ca3...	entry	13	1
79c0f8dbc31024ca3...	requires	13	1
79c0f8dbc31024ca3...	virion surface...	13	1
79c0f8dbc31024ca3...	glycoprotein	13	1
79c0f8dbc31024ca3...	(gp)	13	1
79c0f8dbc31024ca3...	composed	13	1
79c0f8dbc31024ca3...	trimer of	13	1
79c0f8dbc31024ca3...	heterodimers	13	1
79c0f8dbc31024ca3...	(gp1/gp2).	13	1
79c0f8dbc31024ca3...	gp1	13	3
79c0f8dbc31024ca3...	subunit contains	13	1
79c0f8dbc31024ca3...	heavily	13	1
79c0f8dbc31024ca3...	glycosylated	13	1
79c0f8dbc31024ca3...	domains,	13	1
79c0f8dbc31024ca3...	glycan cap	13	1
79c0f8dbc31024ca3...	mucin-like	13	1
79c0f8dbc31024ca3...	domain	13	1

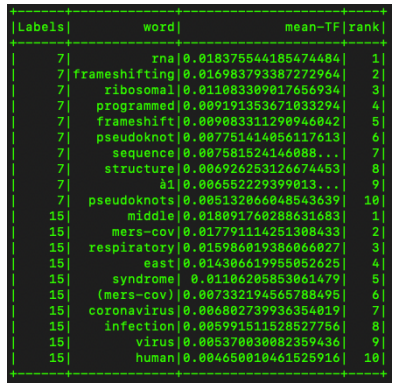
8. **calcolo della TF di ogni termine per ogni paper**: il DF delle occorrenze per ogni parola viene messo prima in join con quello della lunghezza di ogni paper e poi viene aggiunta la colonna T-F come risultato dell'espressione Occurrences/paperLength;

word	T-F	Labels	paper_id
background:	0.016666666666666666	1	27bafe2069fadba96...
elite	0.03333333333333333	1	27bafe2069fadba96...
non-progressors	0.03333333333333333	1	27bafe2069fadba96...
(plasma viral	0.016666666666666666	1	27bafe2069fadba96...
load	0.016666666666666666	1	27bafe2069fadba96...
<50	0.016666666666666666	1	27bafe2069fadba96...
copies/ml	0.016666666666666666	1	27bafe2069fadba96...
antiretroviral<br...	0.016666666666666666	1	27bafe2069fadba96...
constitute	0.016666666666666666	1	27bafe2069fadba96...
tiny	0.016666666666666666	1	27bafe2069fadba96...
fraction	0.016666666666666666	1	27bafe2069fadba96...
hiv-infected i...	0.016666666666666666	1	27bafe2069fadba96...
12	0.03333333333333333	1	27bafe2069fadba96...
follow-up	0.016666666666666666	1	27bafe2069fadba96...
cohort	0.016666666666666666	1	27bafe2069fadba96...
13 long-term	0.016666666666666666	1	27bafe2069fadba96...
(ltnp)	0.016666666666666666	1	27bafe2069fadba96...
identified	0.016666666666666666	1	27bafe2069fadba96...
135 individuals	0.016666666666666666	1	27bafe2069fadba96...
transfusion-acquired	0.016666666666666666	1	27bafe2069fadba96...

9. **calcolo della TF media**: considerando che ci possono essere paper di una categoria che non contengono una determinata parola, il calcolo della media viene eseguito mettendo in join il DF ottenuto al punto 8 e quello ottenuto al punto 3, in modo da non ottenere problemi di counting. Una volta effettuato il join, per calcolare la somma delle T-F viene fatto un `groupBy` per Labels, word, paperPerLabel e viene usata la funzione di aggregazione `sum`. Ne deriva una nuova colonna calcolata tramite l'espressione totalTF/paperPerLabel;

Labels	word	mean-TF
7	require	1.7825311942959E-4
7	heptanucleotide	0.001267791609545222
7	"slippery" seq...	1.7825311942959E-4
7	yyz	1.7825311942959E-4
7	(where	4.002287021154945E-4
7	a,	1.7825311942959E-4
7	u,	3.5650623885918E-4
7	u;	1.7825311942959E-4
7	a, hansjö	1.7825311942959E-4
7	rg	1.7825311942959E-4
7	hauser,	1.7825311942959E-4
7	†	3.5650623885918E-4
7	donald	1.7825311942959E-4
7	m.	1.7825311942959E-4
7	coen*	1.7825311942959E-4
7	c)	1.7825311942959E-4
7	are augmented	1.7825311942959E-4
7	stimulator,	3.810928760218821E-4
7	*department	1.7825311942959E-4
7	of biological	1.7825311942959E-4

10. **raggruppamento dei valori per ogni categoria, ordinandoli e prendendo solo i primi dieci**, ottenuto grazie alle `Window` function di SparkSQL



Labels	word	mean-TF	rank
7	rna	0.018375544185474484	1
7	frameshifting	0.016983793387272964	2
7	ribosomal	0.011083309017656934	3
7	programmed	0.009191353671033294	4
7	frameshift	0.009083311290946042	5
7	pseudoknot	0.00775141405611 7613	6
7	sequence	0.007581524146088...	7
7	structure	0.006926253126674453	8
7	à1	0.006552229399013...	9
7	pseudoknots	0.005132066048543639	10
15	middle	0.018091760288631683	1
15	mers-cov	0.017791114251308433	2
15	respiratory	0.015986019386066027	3
15	east	0.014306619955052625	4
15	syndrome	0.011062058530614 79	5
15	(mers-cov)	0.007332194565788495	6
15	coronavirus	0.006802739936354019	7
15	infection	0.005991511528527756	8
15	virus	0.005370030082359436	9
15	human	0.004650010461525916	10

La history del job SparkSQL che include sia la classifica delle parole con il maggior numero di occorrenze medie che la frequenza media per categoria è raggiungibile al link `http://isi-vclust0.csr.unibo.it:8088/cluster/app/application_1583679666662_3404`.

5 Data visualization

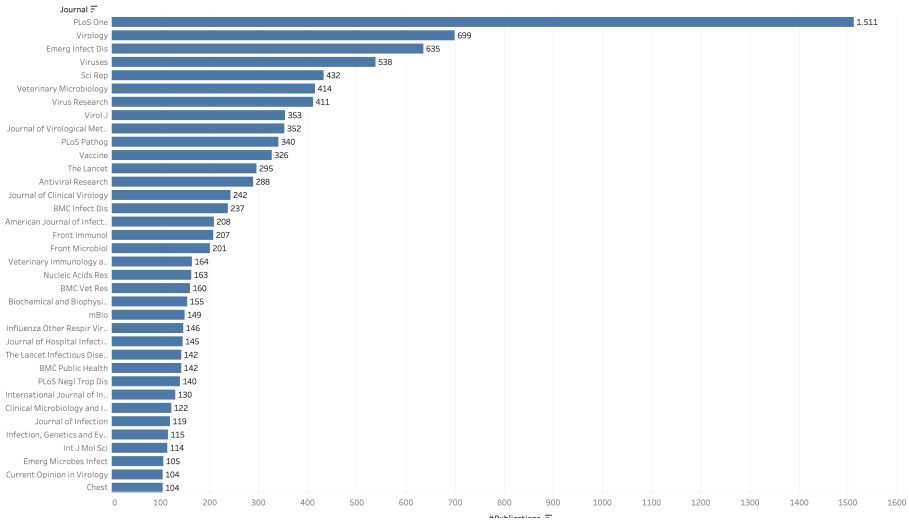


Figure 9: Classifica delle riviste più rilevanti mostrata con un bar chart

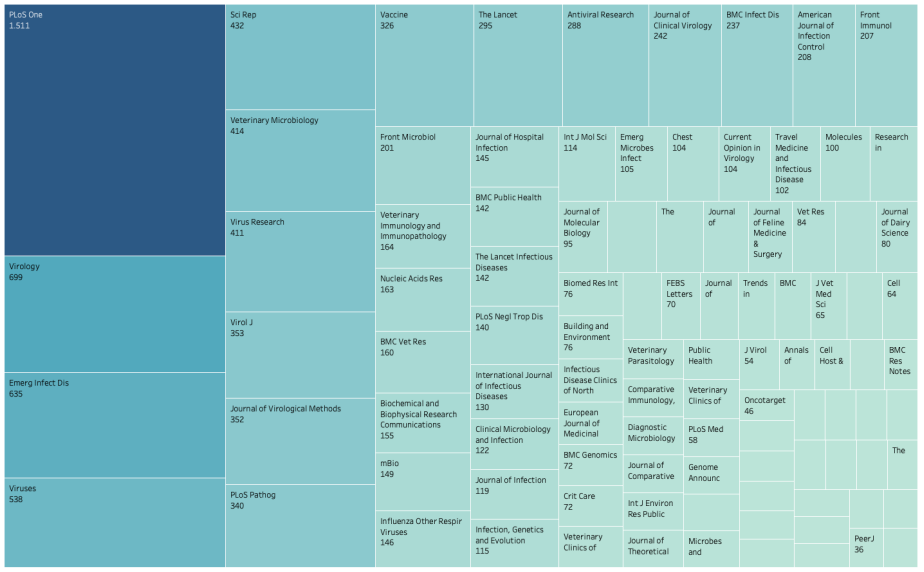


Figure 10: Classifica delle riviste più rilevanti mostrata con una Tree Map

Figure 11: Parole con la più elevata frequenza media per categoria: le parole di uno stesso colore appartengono alla stessa categoria; la parole si distinguono in grandezza in base alla loro frequenza media