Distributed Computing and Storage Architectures

Project MapReduce

Erick Escobar Gallardo

erick.emmanuel.escobar.gallardo@vub.be

Reza Amini

reza.amini@vub.be

December, 2020

Contents

1 Introduction

MapReduce is a technique in which a huge program is subdivided into small tasks, and run paralleled to make our computation faster. In this assignment, we were supposed to do different tasks using MapReduce jobs in Python. For these tasks we will use the MRJob library that lets you write MapReduce jobs in Python and run it in several platforms. Each job in the MRJob library has different steps called MRStep, and each step could have three functions including mapper, combiner and/or reducer. As the job author, you write map, combine, and reduce functions that are submitted to the job tracker in MRJob for execution [3].

- A mapper takes a single key and value as input, and returns zero or more (key, value) pairs. The pairs from all map outputs of a single step are grouped by key.
- A combiner takes a key and a subset of the values for that key as input and returns zero or more (key, value) pairs. Combiners are optimizations that run immediately after each mapper and can be used to decrease total data transfer. Combiners should be idempotent (produce the same output if run multiple times in the job pipeline).
- A reducer takes a key and the complete set of values for that key in the current step, and returns zero or more arbitrary (key, value) pairs as output.

The following sections of the document describes the implementation process of each tasks, and the results obtained by executing each task on the datasets assigned.

Table 1: Contribution

Name	**Tasks**
Erick Escobar	Task 1,2,5
Reza Amini	Task 3,4,6

2 Implementation

In this section we are going to explain further details about the implementation process of each task.

2.1 Task 1: Find the 50 most common keywords for all movies and shorts

The data-set used for this task is "title.basics.tsv" that contains information of movies,tv-shows and shorts from IMDB. The columns description of this data-set are:
[**tconst titleType primaryTitle originalTitle isAdult startYear endYear runtimeMinutes genres**].
The objective of this task is to filter all the entries of the type movie and short and find the top 50 most common keywords used in the primary titles.
In order to do this we will define a **two step job**, consisting in:

1. **Step 1:** First step consisting on a mapper, combiner, and reducer.

   (a) **mapper:** The mapper will split each tabbed word [\t] and will filter and analyze only the words in the third column (*"primaryTitle"*) whose values in the second column (*"titleType"*) is *"movies"* or *"shorts"*. Since the column *"primaryTitle"* contains sentences in different languages, and uses different forms of punctuation, we will proceed to filter the sentences by eliminating punctuation sign, tokenizing the sentences into words, removing the stop-words and eliminating non-alphabetic words. After the filtering process, the mapper will return a tuple for each token in the filtered sentence consisting in: (word, 1).

   (b) **combiner:** Combine the values of a key in a single mapper, it return the sum of values for a single key. In this case will sum all the occurrences of a word in a single mapper, and return a key-value pair: (word, sum(occurrences)).

   (c) **reducer:** Merge all the intermediate counts(values) associated with the same word(key). A reducer will process a same key received from different combiners and will merge (sum) the values of all of those keys. It will return a None key,

and a tuple (sum(counts), word), since the key is None, this means that all the yielded values will end up in a the same process in the next step, in this case a reducer.

2. **Step 2:** This steps consist on only a reducer.

   (a) **reducer:** This process will receive all tuples from last ste. It is important to consider that the default protocol used by jobs to write output and communicate between steps is JSON, and then the received input of this process will be encoded as that [3]. Knowing this the first part will consist in transforming the input into a list of tuples (count,word), and after that the list will be sorted according to the count values in reverse order (highest count value first). Finally, the reducer will yield the 50 first elements of the sorted list.

```
list_counts = []
for count, word in data:
    list_counts.append((count, word))
sorted_list = sorted(list_counts, key=lambda x: x[0], reverse=
   True)
for i in range(50):
    yield sorted_list[i]
```

Listing 1: Filter process

2.2 Task 2: Top 15 keywords for each movie genre

The data-set used for this task is also "title.basics.tsv" that contains information of movies,tv-shows and shorts from IMDB. The columns description of this data-set are:
[**tconst titleType primaryTitle originalTitle isAdult startYear endYear runtimeMinutes genres**].
The objective of this task is to return the top 15 most common key words in the *"primaryTitle"* column for each possible **genre** of the type **movie**.
In order to do this we will define a **two step job**, consisting in:

1. **Step 1:**

   (a) **mapper:** This mapper will split each tabbed word [\t] and will filter and analyze only the words in the third column (*"primaryTitle"*) and last column (*"genres"*) whose value in the second column is *"movie"*. The mapper will yield a tuple (genre,title) for each genre found in the column (*"genres"*). For example the following row:

Table 2: IMDB example data

titleType	**primaryTitle**	**genres**
movie	The Beauty from Roumeli	Drama,Romance

   When processed will yield a tuple (genre,title) for each genre found in the *"genres"* column, in this case it would yield the tuples: ("The Beauty...","Drama"), ("The Beauty...","Romance").

2. **Step 2:**

   (a) **mapper:** This mapper will receive the output tuples from the previous step, and tokenize and filter the words found in the title and return a tuple for each received genre and a dictionary of occurrences of words in the movie title. For example, in the data-set sample show in table 2 would yield a tuple: ("The Beauty from Roumeli","Romance") that will be processed by this mapper and the tuples yielded would have the following form: (key, dictionary), where key is the genre keyword and dictionary contains a dictionary of occurrences of the words found in the movie title:

("drama",{"beauty":1,"from":1,"roumeli":1})

   (b) **combiner:** Combine the dictionaries(counts) that have the same key(genre) in a single mapper. Suppose that a single mapper yields two tuples with the same key:

1. ("drama",{"beauty":1,"from":1,"roumeli":1})
2. ("drama",{"mom":1,"home":1,"alone":1})

The combiner will merge both dictionaries, thanks to the use of the *collections* library from python and the use of the *Counter* method, the result would be:

1. (”drama”,{”beauty”:1,”from”:1,”roumeli”:1,”mom”:1,”home”:1,”alone”:1})

(c) **reducer:** Merge all the intermediate counts dictionaries associated with the same key (genre) and return a key value pair: (key,dict_top_15) where the key is a movie genre and dict_top_15.keys contains the keys of the 15 most frequent words in each movie genre.

```
sj_pro = {}  # accumulated dictionary
for sj in counts:
    sj_pro = Counter(sj_pro) + Counter(sj)
top_15 = sj_pro.most_common(15)  # get the 15 words with the
   highest number of occurrences.
dict_top_15 = dict(top_15)   # translate top_15 from a list to a
   dictionary.
yield (key, list(dict_top_15.keys()))
```

Listing 2: Reducer - Task 2

2.3 Task 3: Top 10 best customers for each retail year

The data-sets used for this task are “retail0910.csv” and “retail1011.csv” that contain information of retail sales of an individual company for 2009-2010 and 2010-2011 respectively. These two data-sets have the same columns description including:
[**Invoice, StockCode, Description, Quantity, InvoiceDate, Price,Customer ID,Country**].
The objective of this task is to find the top 10 customers of each retail year that bought the most in terms of total revenue (quantity * price).
In order to do this we will define a **two step job**, consisting in:

1. **Step 1:**

(a) **mapper:** This mapper splits the received rows from the input to different cells. Then, it extracts customer id, amount which is Quantity * Price and the year of

the invoice. Finally, the mapper returns a set of (key, value) pairs that each key is a combination of customerID and year, and each value is the calculated amount: ((customerID, year), amount).

(b) **combiner:** This combiner sums the amount calculated by the mapper per each key((customerID, year). On the other word, it calculates how much each customer bought in each year. So, it returns a set of (key, value) pairs that each key is a combination of customerID and year, and each value is sum(amount): ((customerID, year), sum(amount)).

(c) **reducer:** It merge all the intermediate amount values associated with the same key (id_year).

2. **Step 2:**

(a) **reducer:** This reducer receives a list of (customerID, totalamount) per year, and it pushs the value to an array. Then, it sorts the array ascendingly based on total amount. Finally, it fetches top 10 from the sorted list. So, the output of this reducer would be 10 (customerID, totalamount) per each year.

2.4 Task 4: Best selling product

In this task, we used exactly the same data-sets as task 3. The objective of this task is to find the best selling product based on total quantity as well as total revenue for both retail years.

In order to do this we will define two jobs with two steps, consisting in:

- Based on total quantity

1. **Step 1:**

(a) **mapper:** This mapper splits the received rows from the input to different cells. Then, it extracts stock which is product id, Quantity and the year of the invoice. Finally, the mapper returns a set of (key, value) pairs that each key is a combination of stock and year, and each value is the quantity: ((stock, year), quantity).

(b) **combiner:** This combiner sums the quantity per each key((stock, year). On the other word, it calculates how many each product is sold in each year. So, it returns a set of (key, value) pairs that each key is a year, and each value is the combination of stock and total quantity: year, (sum(quantity), stock).

(c) **reducer:** It merges all the intermediate quantity values associated with the same key (stock_year).

2. **Step 2:**

(a) **reducer:** This reducer receives a list of (sum(quantity), stock) per year, and it returns the row with the maximum quantity. So, the best selling product based on total quantity in each year will be found.

- Based on total revenue

1. **Step 1:**

(a) **mapper:** This mapper splits the received rows from the input to different cells. Then, it extracts stock which is product id, Quantity, amount, and the year of the invoice. Next, it calculates revenue which is quantity * amount. Finally, the mapper returns a set of (key, value) pairs that each key is a combination of stock and year, and each value is the revenue: ((stock, year), revenue).

(b) **combiner:** This combiner sums the revenue per each key((stock, year). On the other word, it calculates how much each product is sold in each year. So, it returns a set of (key, value) pairs that each key is a year, and each value is the combination of stock and revenue: year, (sum(revenue), stock).

(c) **reducer:** It merges all the intermediate revenue values associated with the same key (stock_year).

2. **Step 2:**

(a) **reducer:** This reducer receives a list of (sum(revenue), stock) per year, and it returns the row with the maximum revenue. So, the bestselling product based on revenue in each year will be found.

2.5 Task 5: Cosine Similarity

The data-set used for this task is a json file called "arxivData.json" that contains entries of 20.000 scientific papers on Arxiv.

The objective of this task is randomly select a paper summary from the input file and calculate the Cosine Similarity score of the summary of the random paper against all the summaries of the papers in the json file. The Cosine Similarity , given two input documents represented as vectors ($\vec{a}$, $\vec{b}$), is represented as [1] :

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2}\sqrt{\Sigma_1^n b_i^2}}$$

where, $\vec{a}\cdot\vec{b} = \Sigma_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

The output this task will return the paper with the highest Cosine Similarity score against the random paper.

In order to implement this task we will define a **two step job**, consisting in:

1. **Step 1:**

   (a) **mapper_raw:** *mapper_raw* is a special method implementation that can receive an entire file as input. The mapper will receive a JSON file as input, read the file and extract the summary of each object and create a bag of words embedding of all the summaries. This mapper will also generate a random number according to the number of rows in the resulting bag of words embedding matrix. The mapper will yield a tuple key-value, and the key consist of a tuple of a summary ID and the generated sample ID, and the value is a tuple of a word embedding representation of a summary and the random summary array representation.

   In order to construct the bag of words embedding model, a single mapper needs to do the training and learning process entirely, that is why we rely in the method *mapper_raw*. Recapping the work done in this mapper:

   - **Vectorize and construct the bag of words embedding model** from all the summaries in the input file, using the method *CountVectorizer* from the library *sklearn*. This method will apply NLP pre-processing task such as

removal of stop-words before constructing the model. We will use the bag of words embedding model to represent the summaries as vectors because it is the simplest implementation but a downside is that the memory needed to construct the model gets bigger as the size of the vocabulary increases.

- **Sample and generate a random number** according to the total number of rows of the generated bag-of-words matrix model (the rows represents the summary and the columns represents each word in the vocabulary).
- **Yield a tuple** ((sumID,randID),(arrSum,randSum)) for each summary in the input file, where sumID is the ID of a summary, randID is the ID of the random choosen summary, arrSum is the bag of words vector representation of a summary and randSum is the bag of words vector representation of the random summary.

(b) **reducer:** This reducer will calculate the cosine distance between two input summaries and yield a tuple: (ID, cosine distance) where ID is a tuple consisting of the ID of a summary and the ID of the random summary (sumID,randID) and *cosineDistance* is the cosine similarity score between arrSum and randSum.

```
def cosine_distance(sumOne, sumTwo):
    """
    Calculates the cosine similarity score between two given
   summaries
    :param sumOne: list containing the word embedding(vector
   representation) of a summary
    :param sumTwo: list containing the word embedding(vector
   representation) of a summary
    :return: returns cosine distance coefficient, indicating how
   close are the summaries.
    """
    dot_product = np.dot(sumOne, sumTwo)
    norm_sumOne = np.linalg.norm(sumOne)
    norm_sumTwo = np.linalg.norm(sumTwo)
    cos = dot_product / (norm_sumOne * norm_sumTwo)
    return cos
```

Listing 3: Cosine Distance

2. **Step 2:**

   (a) **reducer:** Yields the data tuple key-value: (ID, cosineDistance) with highest cosine distance value.

2.6 Task 6: Matrix Multiplication

In this task, we used two provided matrices including matrix A and matrix B. Since these two matrices are too large, and execution of the tasks with these inputs takes a long time, we provided two smaller matrices which matrix A is 2*4 and matrix B is 4*3. The objective of this task is matrix multiplication. So, the output of multiplication is supposed to be matrix C which is 2*3.
In order to do this we will define two step job, consisting in:

1. **Step 1:**

   (a) **mapper:** This mapper splits the received rows from the input to different items. Then it checks the input filename in order to compute each matrix separately. It iterates each input row to calculate the following formula, and finally it returns a set of (key, value) pairs that each key has a list of values [2].
   For Matrix A (k, v)=((i, k), (A, j, Aij)) for all k
   For Matrix B (k, v)=((i, k), (B, j, Bjk)) for all i

   (b) **reducer:** This reducer gets the (key, value) pairs, and then it sorts the list of the values of each key.
   Reducer(k, v)=(i, k)= Make sorted Alist and Blist

2. **Step 2:**

   (a) **mapper:** This mapper gets sorted (key, value) pairs, and then it multiplies the values. It also stores the multiplied values in matrix C.

Reducer(k, v)=(i, k)= Summation (Aij * Bjk)) for j

Output= ((i, k), sum)

3 Results

All the task were tested and implemented in Python 3 version 3.7.9. In this section we will explain how to run each one of the task and mention the external libraries used for the implementation if needed.

3.1 Task 1

In order to run the Task 1, the following command should be executed.

```
python task_1.py ../1_IMDB/title.basics.tsv
```

In the second argument of the command (../1_IMDB/title.basics.tsv) you should pass the location of the file *title.basics.tsv*.

The external library NLTK was used in the implementation of this task. In order to install the library using *pip*, execute the following command:

```
pip install --user -U nltk
```

The results obtained are shown in the following listing, the list represent the 50 most common used keywords.

```
14474   "love"
9263    "life"
7231    "story"
6642    "night"
6516    "last"
6407    "day"
5653    "time"
5179    "girl"
```

```
4452    "little"
4418    "new"
4377    "black"
4333    "world"
4154    "home"
3762    "death"
3600    "film"
3597    "dead"
3558    "red"
3551    "house"
3548    "two"
3434    "city"
3375    "lost"
3253    "movie"
3119    "good"
3071    "dream"
3005    "heart"
2984    "first"
2910    "project"
2872    "big"
2838    "boy"
2824    "white"
2796    "road"
2725    "blue"
2700    "dark"
2685    "blood"
2655    "days"
2617    "way"
2545    "vs"
2523    "mr"
2518    "woman"
2509    "dont"
2505    "american"
2459    "untitled"
2447    "girls"
2412    "like"
```

```
2362    "three"
2320    "back"
2281    "king"
2235    "game"
2233    "live"
2221    "great"
```

3.2 Task 2

In order to run the Task 2, the following command should be executed.

```
python task_2.py ../1_IMDB/title.basics.tsv
```

In the second argument of the command (../1_IMDB/title.basics.tsv) you should pass the location of the file.

The results of this task are shown in the following listing:

```
"Action"        ["black", "ang", "last", "movie", "ng", "blood", "death",
   "dragon", "dead", "ka", "project", "untitled", "vs", "red", "ninja"]
"Adult" ["sex", "love", "girls", "hot", "black", "girl", "erotica", "lust
   ", "night", "blue", "swedish", "erotic", "young", "big", "sweet"]
"Adventure"     ["movie", "adventures", "last", "lost", "island", "
   adventure", "black", "king", "legend", "world", "treasure", "jungle", "
   project", "journey", "wild"]
"Animation"     ["movie", "little", "adventures", "adventure", "legend", "
   dragon", "magic", "king", "world", "space", "big", "untitled", "story",
    "life", "christmas"]
"Biography"     ["story", "life", "untitled", "project", "love", "american
   ", "world", "last", "journey", "king", "art", "time", "john", "legend",
    "film"]
"Comedy"        ["love", "movie", "life", "girl", "night", "little", "mr",
    "big", "girls", "story", "last", "dead", "time", "day", "two"]
"Crime" ["night", "murder", "black", "crime", "love", "death", "city", "
   story", "last", "dead", "girl", "killer", "kill", "life", "blood"]
```

```
"Documentary"   ["story", "life", "world", "love", "documentary", "journey
   ", "last", "american", "new", "film", "art", "time", "home", "road", "
   city"]
"Drama" ["love", "life", "story", "last", "girl", "night", "woman", "black
   ", "day", "time", "little", "heart", "days", "untitled", "project"]
"Family"        ["christmas", "little", "love", "life", "story", "movie",
   "family", "home", "magic", "boy", "adventures", "world", "time", "
   secret", "kids"]
"Fantasy"       ["magic", "love", "legend", "time", "ghost", "movie", "
   last", "world", "night", "little", "dream", "story", "dragon", "life",
   "dead"]
"Film-Noir"     ["night", "woman", "big", "house", "city", "street", "
   murder", "strange", "story", "lady", "dark", "johnny", "danger", "fear
   ", "black"]
"Game-Show"     ["wait", "circus", "untitled", "poker", "project", "six",
   "tape", "show", "citychase", "dont", "tell", "live", "schlag", "tier",
   "alien"]
"History"       ["story", "last", "life", "american", "love", "king", "
   black", "world", "history", "time", "city", "road", "journey", "great",
    "land"]
"Horror"        ["dead", "night", "blood", "house", "dark", "death", "
   ghost", "horror", "evil", "zombie", "black", "hell", "killer", "vampire
   ", "terror"]
"Music" ["live", "music", "story", "rock", "love", "life", "concert", "
   song", "world", "dance", "night", "movie", "time", "new", "band"]
"Musical"       ["love", "musical", "song", "girl", "music", "dance", "
   life", "live", "amor", "rock", "night", "story", "broadway", "paris", "
   little"]
"Mystery"       ["murder", "night", "mystery", "dark", "black", "house", "
   love", "death", "last", "dead", "secret", "case", "girl", "midnight", "
   ghost"]
"News"  ["story", "american", "life", "documentary", "world", "city", "new
   ", "land", "america", "water", "people", "home", "time", "love", "art"]
"Reality-TV"    ["volume", "world", "film", "return", "house", "movie", "
   music", "festival", "girls", "gone", "wild", "independent", "part", "
   dead", "christ"]
```

```
"Romance"       ["love", "girl", "life", "story", "night", "woman", "last
   ", "heart", "time", "romance", "little", "day", "lady", "two", "summer
   "]
"Sci-Fi"        ["space", "time", "project", "alien", "world", "star", "
   last", "untitled", "earth", "dark", "planet", "movie", "black", "dead",
    "moon"]
"Short" ["kampen", "hj\u00e4rta", "blott", "dr\u00f6m", "qila", "erbo", "
   fire", "island", "kids", "s\u00e4terj\u00e4ntan", "mothers", "promise",
    "barack", "obama", "bio"]
"Sport" ["vs", "liverpool", "football", "club", "city", "united", "
   manchester", "fc", "story", "chelsea", "game", "west", "world", "
   tottenham", "hotspur"]
"Talk-Show"     ["great", "debate", "sherlock", "holmes", "talk", "psychic
   ", "podcast", "behind", "new", "world", "show", "tv", "stars", "special
   ", "interview"]
"Thriller"      ["night", "dark", "dead", "black", "last", "blood", "death
   ", "house", "love", "untitled", "project", "killer", "kill", "red", "
   day"]
"War"   ["battle", "last", "story", "soldier", "soldiers", "love", "hell",
    "red", "operation", "life", "home", "heroes", "days", "land", "death"]
"Western"       ["west", "trail", "kid", "gun", "wild", "texas", "law", "
   cowboy", "last", "range", "gold", "western", "riders", "rider", "
   fighting"]
"\\N"   ["love", "onna", "chikan", "sex", "wa", "life", "hitozuma", "girl
   ", "densha", "live", "night", "girls", "untitled", "story", "b\u00f4k\
   u00f4"]
```

3.3 Task 3

In order to run the Task 3, the following command should be executed.

```
python task_3.py ../2_RETAIL/retail0910.csv ../2_RETAIL/retail1011.csv
```

In the second and third arguments of the command, you should pass the location of the files. The results of this task are shown in the following listing:

```
"2009"  [["", 136575.05999999706], ["18102", 41005.73999999999], ["13694",
    20427.90999999999], ["14646", 14696.47], ["15061",
   14312.539999999999], ["14156", 11024.719999999998], ["15311",
   9293.389999999994], ["17511", 8367.809999999996], ["17850",
   7411.139999999996
], ["13777", 7125.04]]
"2010"  [["", 1249054.000000027], ["18102", 328605.6], ["14646",
   237748.46000000008], ["14156", 172470.53000000003], ["14911",
   138259.1000000001], ["13694", 110629.54999999997], ["15061", 87800.86],
    ["17511", 82027.54999999999], ["16684", 75837.25], ["13089", 58565
.57]]
"2011"  [["", 1253329.1200000215], ["14646", 270897.13999999996],
   ["18102", 228603.87999999995], ["17450", 185453.33], ["14911",
   125815.48999999996], ["12415", 123725.45], ["14156",
   113069.44000000005], ["17511", 81490.62000000001], ["16684", 62778.68],
    ["13694", 5
9768.06]]
```

3.4 Task 4

In order to run the Task 4, the following commands should be executed which the first one is based on total quantity, and the second one is based on total revenue.

```
python task_4_a.py ../2_RETAIL/retail0910.csv ../2_RETAIL/retail1011.csv
```

```
python task_4_b.py ../2_RETAIL/retail0910.csv ../2_RETAIL/retail1011.csv
```

In the second and third arguments of the commands, you should pass the location of the files.

The results obtained are shown in the following listings. This list represents the best selling product based on total quantity in each year. In each row, the first part is the total quantity, and the second one is stockcode of the product beside the year.

```
6211    ["85123A", "2009"]
58553   ["21212", "2010"]
53719   ["22197", "2011"]
```

This list represents the best selling product based on total revenue in each year. In each row, the first part is the total revenue, and the second one is stockcode of the product beside the year.

```
18574.579999999998      ["DOT", "2009"]
189948.81999999992      ["22423", "2010"]
181574.28999999998      ["DOT", "2011"]
```

3.5 Task 5

In order to run the Task 5, the following command should be executed.

```
python task_5 ../3_TEXT-SIMILARITY/arxivData.json
```

In the second argument of the command (../3_TEXT-SIMILARITY/arxivData.json) you should pass the location of the file.
In order to implement this task the following external libraries were used: SKLearn and Numpy.

```
pip install -U scikit-learn
```

```
pip install numpy
```

The result obtained by executing this task is the following:

```
["quant-ph/9907009v2", "1511.05720v1"]  0.010040241611281236
```

Where the first tuple shows the ID of two summaries, the first ID (”quant-ph/9907009v2) represents the summary ID of the summary that returned the highest cosine distance in

relation to the random selected summary (the second ID in the tuple "1511.05720v1"). The value represents the cosine distance.

3.6 Task 6

In order to run the Task 6, the following command should be executed.

```
python task_6.py ../4_MATRIX/A.txt ../4_MATRIX/B.txt
```

In the second and third arguments of the command, you should pass the location of the files. The sample inputs we passed as two matrices are as below:

```
1 2 3 4
5 6 7 8
```

```
10 11 12
13 14 15
16 17 18
19 20 21
```

Then, the result obtained by executing this task is the following:

```
160.0 170.0 180.0
392.0 418.0 444.0
```

4 Conclusions

By using MapReduce, we can write applications to process huge amounts of data, in parallel and in a reliable manner. MapReduce framework operates using three important methods including mapper, combiner and reducer. Using the MapReduce technique and a hadoop implementation (or any other framework) we could process big data files using the MapReduce techniques and divide the load of each task into different nodes(workers) that will perform different tasks.

Bibliography

[1] Anna Huang. "Similarity measures for text document clustering". In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference* (Jan. 2008).

[2] *Matrix Multiplication With 1 MapReduce Step*. URL: https://www.geeksforgeeks.org/matrix-multiplication-with-1-mapreduce-step.

[3] *mrjob v0.7.4 documentation*. URL: https://mrjob.readthedocs.io/en/latest/protocols.html.