CSE 587 - Data Intensive Computing

Assignment 2: Big Data Processing with Hadoop

Submitted by:

Vishva Nitin Patel (vishvani@buffalo.edu)

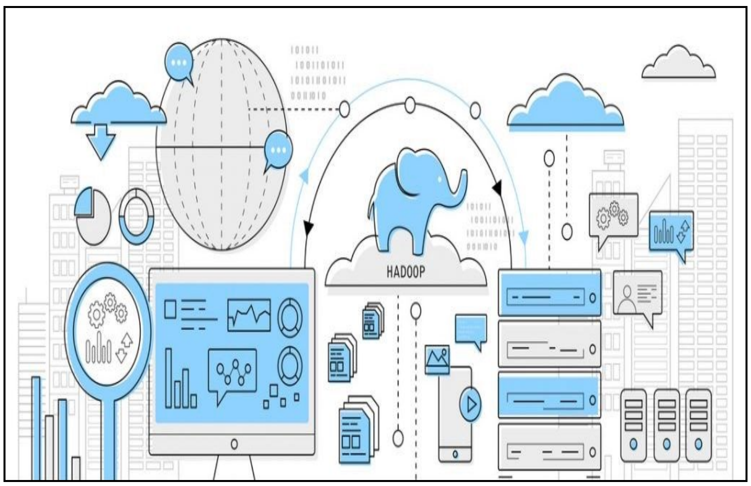
Yash Nitin Mantri (yashniti@buffalo.edu)

Abhishek Muni (amuni@buffalo.edu)

Guided By:

Deen Dayal Mohan

(Teaching Assistant - CSE Department, UB)



Introduction

This project aims to introduce the basic setup of the Hadoop Distributed File System along with a hands-on approach to understanding the Map-Reduce paradigm of implementing algorithms. The tasks learnt include - 1. Map-Reduce algorithm to count the number of words in a text file (This part introduces us to the basics of writing an elementary map-reduce program), 2. Implementation of map reduce algorithm to form modified trigrams around given keywords and find the 10 most frequently such occuring trigrams. 3. Map-Reduce algorithm to produce the inverted index result for the gutenberg dataset, 4. Perform Relational Join using Map-Reduce algorithm to join the given two datasets using a primary key. and 5. Implementing KNN Algorithm using the Map-Reduce paradigm.

PART -1: Setup and Word Count

- **Setup:**

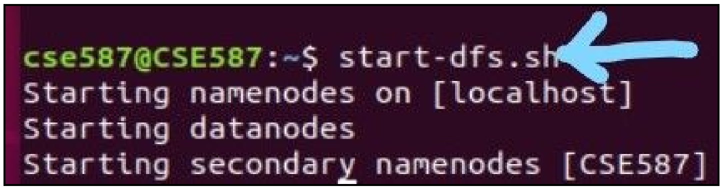


Fig- 1: Starting the distributed file system. This command will start the namenode as well as the data nodes as clusters.

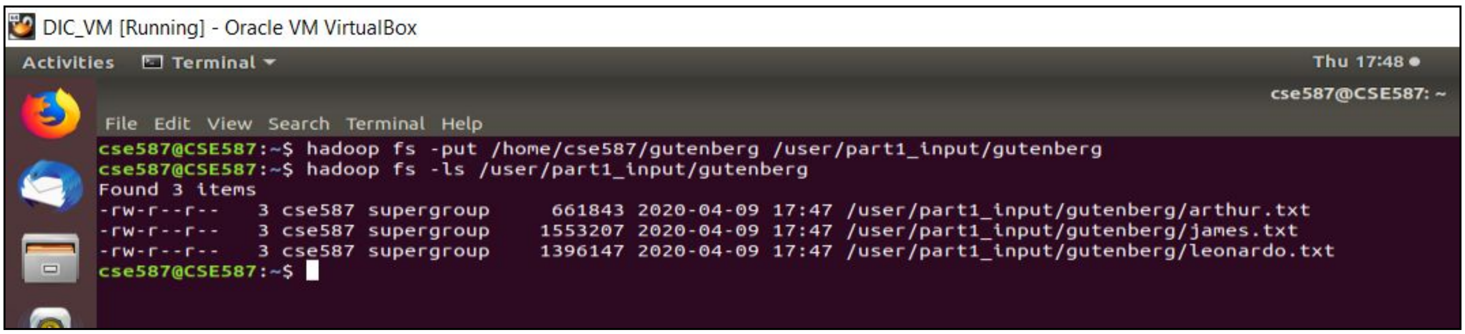


Fig-2: Copying data from local file system to HDFS (gutenberg folder).

- **Implementing a MapReduce algorithm to produce count of every word in the document:**

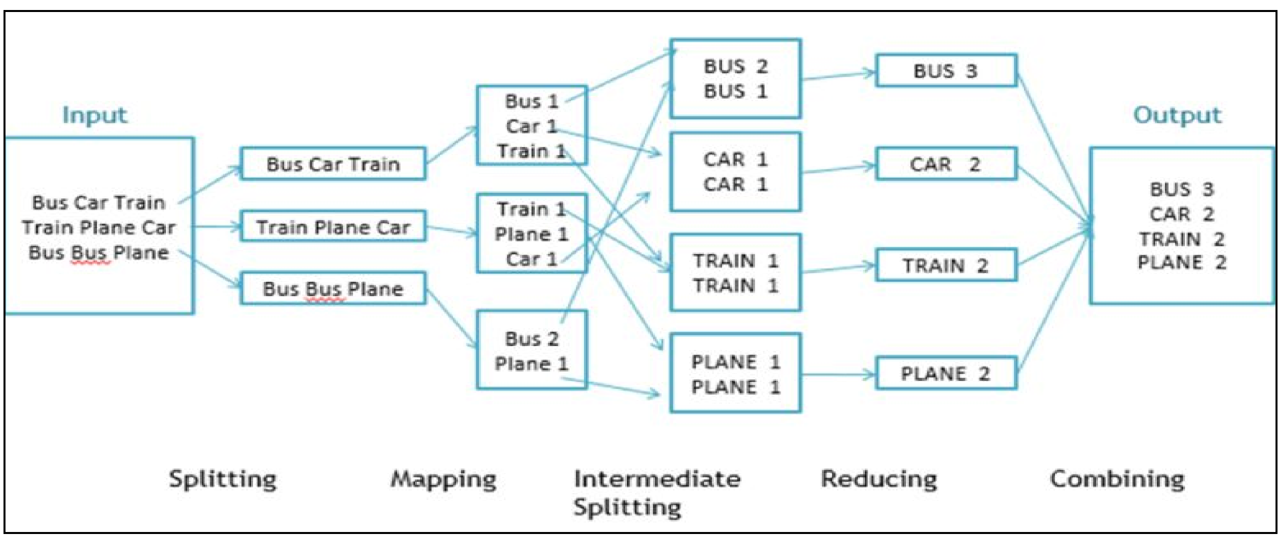


Fig-3: Workflow of the Map Reduce word count algorithm.

```
['"At', 1]
['"BOILING"', 2]
['"B_', 1]
['"Batesian"', 1]
['"Bathers', 1]
['"Beta', 2]
['"Beta"', 1]
['"Big', 1]
['"Bononiae', 1]
['"But', 1]
['"By', 1]
['"Cave-men"', 2]
['"Clean', 1]
['"Come', 1]
['"Cromagnards"', 1]
['"Crookes', 2]
['"DARWIN\'S', 2]
```

Now, as shown in Fig. 4 here, the final result that the reducer generates represents the word-count of each word - the mapper which is the first step creates the <word, 1> pairs - where 1 is statically assigned to each word - at the end it is the reducer that sums up these 1's for each word to generate the "sum-total" word count.

Fig - 4: Output - Resulting from the reducer.

<u>Part-2: Map-reduce implementation to find modified trigram around a keyword:</u>

- **<u>Theory</u>:** Trigrams are sequences of 3 continuous words or other tokens in a string dataset. It is a part of n-gram statistical models which is the building block for information retrieval and natural language processing.
- **<u>Objective</u>:** The goal of our assignment is to implement a map-reduce algorithm to form modified trigrams around the keywords **'science'**, **'sea'** and **'fire' gutenberg dataset.** The mapper should produce the desired trigrams after splitting the content of the text and using these as input the reducer should return the 10 most frequently occuring trigrams in the dataset.
- **<u>Implementation:</u>**
    1. First the input guttenberg file having three different text folders were added to the hadoop system.
    2. Next the data from three the input folders were being input to our mapper and reducers as shown in Fig3. The initial set of mappers takes the scripts from the input folders and modifies it into all possible sets of trigrams after preprocessing them and splitting the lines into words. The output from these mappers are being taken as input for the reducers which finds the total count of the trigrams in each set of the clusters.

of_the_$	113
the_$_and	52
in_the_$	34
from_the_$	33
to_the_$	27
the_open_$	25
$_and_the	17
the_$_to	16
the_$_of	13
by_the_$	12

3. Then finally the output from step2 was feeded to another set of mappers and reducers which finally aggregates the overall count of the trigrams and returns the 10 most frequently ocuuring ones.The final output obtained after this was as follows.

PART -3: Inverted Index

- **Theory:** Inverted Index is a database index storing a mapping from content, such as words or numbers, to its locations in a table, or in a document or a set of documents.
- **Setup:**

```
cse587@CSE587: ~/Desktop/project2/part3
File Edit View Search Terminal Help
cse587@CSE587:~/Desktop/project2/part3$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [CSE587]
cse587@CSE587:~/Desktop/project2/part3$
```

- **Transfer files from local directory to Hadoop:**

```
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -mkdir /home/cse587/project2/
part3/input
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -copyFromLocal /home/cse587/D
esktop/project2/part3/input/*.txt /home/cse587/project2/part3/input
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -ls /home/cse587/project2/par
t3/input
Found 3 items
-rw-r--r--   3 cse587 supergroup     661843 2020-04-18 21:21 /home/cse587/proje
ct2/part3/input/arthur.txt
-rw-r--r--   3 cse587 supergroup    1553207 2020-04-18 21:21 /home/cse587/proje
ct2/part3/input/james.txt
-rw-r--r--   3 cse587 supergroup    1396147 2020-04-18 21:21 /home/cse587/proje
ct2/part3/input/leonardo.txt
cse587@CSE587:~/Desktop/project2/part3$
```

- **Objective:** For this task, we have 3 text files (gutenberg dataset) and the objective is to produce inverted index using Map-Reduce algorithm.
- **Working of the script:**
- In the mapper file, we read the contents of all the text files. Also, we use nltk (Natural Language Toolkit) library to get the list of stopwords.
- **For each line in the mapper:**
  a) Split all the words
  b) Get the file name for each word (Read the path using 'os' library and get the required file name from the path)
  c) Ignore the stop words and give word and file_name as output.
- In the reducer file, we read the output of the mapper file as input.

```
["academy'"]	['leonardo.txt']
["accidental'"]	['arthur.txt']
["acqueducts'"]	['leonardo.txt']
["activity's"]	['arthur.txt']
["aerea'"]	['leonardo.txt']
["aeroplane's"]	['arthur.txt']
["alberti's"]	['leonardo.txt']
["alfieri's"]	['leonardo.txt']
["all'"]	['leonardo.txt']
["all'incontro"]	['leonardo.txt']
["all'uso"]	['leonardo.txt']
["ambrose's"]	['leonardo.txt']
["amoretti's"]	['leonardo.txt']
["andrea's"]	['leonardo.txt']
["angel's"]	['leonardo.txt']
["angelo's"]	['leonardo.txt']
["angler's"]	['arthur.txt']
["animal's"]	['arthur.txt']
["animals'"]	['leonardo.txt']
["another's"]	['arthur.txt']
["ape's"]	['arthur.txt']
["apostles'"]	['leonardo.txt']
["archaeologia'"]	['leonardo.txt']
["archimedes'"]	['leonardo.txt']
["architect's"]	['leonardo.txt']
```

We perform the following steps in reducer:

a) We create a dictionary (output_data) for all the words and filename combinations to give the final result.

b) Here, the word would be the key of the dictionary and file_name would be the value that would be appended for that word.

c) While iterating we check if the key for the word is already created. If not, we create a key for that word.

d) Then, we also check if the file_name is already assigned as value to the given key (word). If not, we append the file_name for the given key.

e) Finally, we iterate the dictionary and produce the reducer output based on the word and its corresponding filename(s).

```
["giorgio's"]	['leonardo.txt']
["giovan'"]	['leonardo.txt']
["giovan'antonio"]	['leonardo.txt']
["giraffe's"]	['arthur.txt']
["giuliano's"]	['leonardo.txt']
["god's"]	['leonardo.txt', 'arthur.txt']
["goethe's"]	['arthur.txt']
["gorilla's"]	['arthur.txt']
["hare's"]	['arthur.txt']
["hart's"]	['leonardo.txt']
["head's"]	['leonardo.txt']
["health'"]	['leonardo.txt']
["highness'"]	['leonardo.txt']
["hornbill's"]	['arthur.txt']
["horse's"]	['arthur.txt', 'leonardo.txt']
["host's"]	['arthur.txt']
["hund'"]	['leonardo.txt']
["i'"]	['leonardo.txt']
["indaco's"]	['leonardo.txt']
["india'"]	['leonardo.txt']
["infant's"]	['arthur.txt']
["ingersoll's"]	['arthur.txt']
["inhabitants'"]	['leonardo.txt']
["insect's"]	['arthur.txt']
["isn't"]	['arthur.txt']
```

PART -4: Relational Join

- **Theory:** Relational Join is a means of combining two or more files (tables) using one or more common columns.
- **Transfer files from local directory to Hadoop:**

```
File  Edit  View  Search  Terminal  Help
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -mkdir /home/cse587/project2/
part4/input
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -copyFromLocal /home/cse587/D
esktop/project2/part4/input/*.csv /home/cse587/project2/part4/input
cse587@CSE587:~/Desktop/project2/part3$ hadoop fs -ls /home/cse587/project2/par
t4/input
Found 2 items
-rw-r--r--   3 cse587 supergroup       3003 2020-04-18 21:49 /home/cse587/proje
ct2/part4/input/join1.csv
-rw-r--r--   3 cse587 supergroup       4837 2020-04-18 21:49 /home/cse587/proje
ct2/part4/input/join2.csv
cse587@CSE587:~/Desktop/project2/part3$
```

- **Objective:** For this task, we convert the 2 excel files into csv files. The objective is to perform a relational join on the 2 csv files and generate a resultant file which contains the output after performing the join.

- **Working of the script:**
  - Here, we consider EmployeeID as the primary key.
  - In the mapper file, we read the contents of both the files and pass them to reducer.
  - In the reducer file, we read the output of the mapper file as input.
  - We perform the following steps in reducer:
    a) In the first file (join1.csv), we have just 2 columns (EmployeeId and name). In the second file (join2.csv), we have 4 columns (EmployeeId, salary, country and passcode).
    b) So, we differentiate them in the reducer by reading the value of the column salary. If the salary is zero ('0'), we can say the row belongs to file 1, otherwise, it belongs to file 2.
    c) Then, we iterate the results of output again and merge them using EmployeeID (primary key).

```
Employee_ID Name    Salary  Country Passcode
16001018 -0115  Sean Herrera    "$28,689"   Guatemala   JFH58LTF7LS
16020401 -5051  Kaitlin Hubbard "$33,868"   Kazakhstan  FAK20ZLP2XX
16030503 -6774  William Maldonado   "$92,019"   Samoa   SBN74FYH6JJ
16030612 -9305  Brennan Boyd    "$65,670"   Haiti   DRK47IOB8ZU
16031211 -8540  Xanthus Roman   "$72,771"   Comoros PAT27JJA6XB
16040110 -9038  Brielle Weiss   "$19,785"   Bolivia JEQ68XYP3MN
16040828 -9221  Dalton Mccall   "$89,983"   "Virgin Islands, United States" KXY23CHX2OC
16050728 -1673  Lunea Clarke    "$72,063"   Chad    RLJ79WSK7XO
16051003 -3665  Colette Kirby   "$41,630"   Dominican Republic  WRN14VLN3VD
16060113 -5817  Ferdinand Stewart   "$92,932"   Monaco  WOV44LRT2ZD
16060415 -7529  Fredericka Davidson "$71,823"   Rwanda  AMK92WUZ6MP
16070128 -6445  Cleo Alvarez    "$42,257"   Panama  QPY55RSZ3WO
16070214 -4304  Deanna Cardenas "$29,284"   Saint Barth�lemy    LLX37SJF7HT
16080512 -1522  Nehru Rivers    "$17,611"   Bhutan  MMY82OHH0QW
16100501 -2636  Basia Ballard   "$01,702"   Mexico  ITO91QFT9AO
16101124 -9891  Kaseem Dickerson    "$75,105"   Turks and Caicos Islands    TWI40PVM4LC
16120428 -6379  Donna Haney "$82,026"   Guinea-Bissau   IVR42SIN5ZY
16140530 -1837  Clark Wall  "$68,573"   Montenegro  XGS23DFI6BM
16160230 -4113  Cecilia Cox "$26,126"   "Virgin Islands, United States" CYF43NLU7YB
```

	A	B	C	D	E
1	Employee ID	Name	Salary	Country	Passcode
2	16001018 -0115	Sean Herrera	$28,689.00	Guatemala	JFH58LTF7LS
3	16020401 -5051	Kaitlin Hubbard	$33,868.00	Kazakhstan	FAK20ZLP2XX
4	16030503 -6774	William Maldonado	$92,019.00	Samoa	SBN74FYH6JJ
5	16030612 -9305	Brennan Boyd	$65,670.00	Haiti	DRK47IOB8ZU
6	16031211 -8540	Xanthus Roman	$72,771.00	Comoros	PAT27JJA6XB
7	16040110 -9038	Brielle Weiss	$19,785.00	Bolivia	JEQ68XYP3MN
8	16040828 -9221	Dalton Mccall	$89,983.00	Virgin Islands, United States	KXY23CHX2OC
9	16050728 -1673	Lunea Clarke	$72,063.00	Chad	RLJ79WSK7XO
10	16051003 -3665	Colette Kirby	$41,630.00	Dominican Republic	WRN14VLN3VD
11	16060113 -5817	Ferdinand Stewart	$92,932.00	Monaco	WOV44LRT2ZD
12	16060415 -7529	Fredericka Davidson	$71,823.00	Rwanda	AMK92WUZ6MP
13	16070128 -6445	Cleo Alvarez	$42,257.00	Panama	QPY55RSZ3WO
14	16070214 -4304	Deanna Cardenas	$29,284.00	Saint Barth�lemy	LLX37SJF7HT
15	16080512 -1522	Nehru Rivers	$17,611.00	Bhutan	MMY82OHH0QW
16	16100501 -2636	Basia Ballard	$1,702.00	Mexico	ITO91QFT9AO
17	16101124 -9891	Kaseem Dickerson	$75,105.00	Turks and Caicos Islands	TWI40PVM4LC

PART - 5 (Bonus): Implementing KNN algorithm using MapReduce:

Ref. https://ijssst.info/Vol-15/No-3/data/3857a513.pdf

The approach and corresponding algorithms can be summarized as follows:

Prior to implementing the K-Nearest Neighbor technique lets see how to handle the input and output for the implementation. Since we are using the MapReduce paradigm we must make sure that the input is in the form of a pair.

The Map routine performs the function of calculating the distance of each data point with the classes and lists it out.

The Reduce routine then chooses the first 'k' Neighbors in increasing order of distances and conducts a majority vote. After which it sets the data point's label as the label of the class with the majority vote count. Now, we need to organize the input and output directories.

To do this let us name the directory that holds the data vectors as vectors and the training and testing data as trainFile and testFile. Having organized our input/output directories and training and testing data ready, we can apply the k-Nearest Neighbor technique in a distributed environment by following the algorithms discussed below to design the Map and the Reduce functions for the k-Nearest Neighbor Technique.

Algorithm 1 Mapper design for k-NN

procedure K-NN MAPDESIGN
- Create list to maintain data points in the testing data-set
- $testList = new\ testList$
- Load file containing testing data-set
- $load\ testFile$
- Update list with data points from file
- $testList <= testFile$
- Open file containing training data set
- $OPEN\ trainFile$
- Load training data points one at a time and compute distance with every testing data point
- $distance\ (trainData, testData)$
- Write the distance of test data points from all the training data points with their respective class labels in ascending order of distances
- $testFile <= testData(dist, label)$
- Call Reducer

end procedure=0

Algorithm 2 Reducer design for k-NN

procedure K-NN REDUCEDESIGN
- Load the value of 'k'
- Load testFile
- $OPEN\ testFile$
- Load test data points one at a time
- $READ\ testDataPoint$
- Initialize counters for all class labels
- $SET\ counters\ to\ ZERO$
- Look through top 'k' distance for the respective test data point and increment the corresponding class label counter
- $for\ i = 0\ to\ k$
- $COUNTER_i + +$
- Assign the class label with the highest count for the testDataPoint in question
- $testDataPoint = classLabel(COUNTER_{max})$
- Update output file with classified test data point
- $outFile = outFile + testDataPoint$

end procedure=0

Algorithm 3 Implementing kNN Function

```
procedure KNN FUNCTION
    Read the value of 'k'
    SET 'k'
    Set paths for training and testing data directories
    SET trainFile
    SET testFile
    Create new JOB
    SET MAPPER to map class defined
    SET REDUCER to reduce class define
    Set paths for output directory
    SUBMIT JOB
end procedure=0
```

0	8
1	1
2	7
3	7
4	6
5	3
6	4
7	2
8	1
9	11
10	3
11	6
12	4
13	8
14	5

← This is the output of our map-reduce based KNN Algorithm for the corresponding Train and Test data provided for the assignment.