Group Members:

Vidhi Jain 2014A7TS0113P
Abhishek Gupta 2014A7PS0026P
Supriya Agarwal 2014A7PS0013P

Lab Evaluation 2 - Mapreduce Project

We are using Apache Spark (Python API - **PySpark**) for implementing the map-reduce programs.

Some important details about <u>Apache Spark</u>:

1. Apache Spark is a fast and *general-purpose cluster computing system*. It provides *high-level APIs in Scala, Java, and Python* that make parallel jobs easy to write, and an optimized engine that supports general computation graphs. [1]
2. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. [2]
3. RDDs support two types of operations: *transformations* **(map)**, which create a new dataset from an existing one, and *actions* **(reduce)**, which return a value to the driver program after running a computation on the dataset. [2]

Following briefly explains the <u>conventions</u> and <u>semantics of the common functions</u> used in our source codes:-

<u>Conventions</u>:

1. sc - SparkContext Object
   Main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs and broadcast variables on that cluster.
2. df - RDD object (explained in point 2 above)
3. Explain the convention of variable names

<u>Functions:-</u>

1. <u>textFile</u> - Function used for reading the data file
   Read a text file from HDFS, a local file system (available on all nodes), or any Hadoop-supported file system URI, and return it as an RDD of Strings.[3]
   Each row in a CSV file will be read as a string. So we will have a collection of strings.
2. <u>map</u> - function used on a RDD object to transform it to an other RDD object, each element in original RDD maps to exactly one element in the transformed RDD
   We provide a function to describe the transformation. We have used anonymous functions (lambda functions in python) for most of the places to describe the transformations.
3. <u>flatMap</u> - similar to map, but each element in original RDD can map to any number of intermediate key-value pairs
4. <u>reduceByKey</u>
   Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce.

5. mapreduce_join(rdd1, rdd2) This is our custom designed implementation of join() operation in pyspark module through map() and reduce() operation paradigm.
We explain our implementation of mapreduce_join using the following e.g.:-
rdd1 - [(k1, v1), (k2, v2), (k2, v3)]
rdd2 - [(k2, v4), (k3, v5)]
We wish to join rdd1 with rdd2 on the basis of keys of the two RDDs, i.e., [k1, k2] from rdd1, and [k2, k3] from rdd2.
We append recordID in the values of RDD, for this we apply a map operation on both RDDs, to change value *v* to *[(recordID, v)]*
rdd1 - [(k1, [(1 , v1)]), (k2, [(1, v2)]), (k2, [(1, v3)])]
rdd2 - [(k2, [(2, v4)]), (k3, [(2, v5)])]
We take union operation on rdd1 and rdd2 to get combined_rdd
combined_rdd = [(k1, [(1 , v1)]), (k2, [(1, v2)]), (k2, [(1, v3)]), (k2, [(2, v4)]), (k3, [(2, v5)])]

We apply reduce operation, collecting the values of corresponding to a key in a list
After reduce
combined_rdd = [(k1, [(1 , v1)]), (k2, [(1, v2), (1, v3), (2, v4)]), (k3, [(2, v5)])]

We apply map on this combined_rdd taking transforming each (k, v) by the following function:-
```
mapreduce_join_util(x):
    first_d = []
    second_d = []
    for u, v in x[1]:
        if u == 1:
            first_d.append(v)
        else:
            second_d.append(v)
    return [(u, v) for u in first_d for v in second_d]
```

x is the key-value pair such as [(k1,[(1, v1)]]. This operation separates the actual values on the basis of record type (indicated by 1 or 2) into two separate lists and then combines them as a list by pairing each element of first list with every element of second list.
The final step is to combine elements from different lists (corresponding to different (key, value) in combined_rdd) into a single list. (flatMap() does it for us :))

Returns an RDD containing all pairs of elements with matching keys in self and other.

Explanation of Map Reduce functions for solving queries on the given dataset:-

1. **List the Top Ten movies according to their highest average rating.**
   Source Code File Name: **Question1.ipynb**
   The following pseudo code explains the series of map and reduce function usages in our code.
   a. Read into df #df - collection of strings, a row in the CSV file is a single string
   b. Transform df into column_data, each string is converted into list of items.
      E.g. “1, Rock on!, 2 ,3” is converted to [“1”, “Rock on!”, “2”, “3”], using map function.
   c. Transform column_data to get a map such that movieid is the key, and a tuple of two items (rating of the movie, and integer literal 1) as value.
      Output Sample:- [('97', (4.0, 1)), ('180', (3.0, 1)), ('97', (2.0, 1))]
   d. Apply reduceByKey operating, all values corresponding to a key are combined using an associative binary function provided by us.
      In our reduce function (lambda x, y: (x[0]+y[0],x[1]+y[1])), x[0]+y[0] returns the rating sum, x[1] + y[1] returns the count of the movies of which sum is calculated.
   e. We again apply transformation (map) operating to change the (movie_rating_sum, movie_count) in value to average movie_rating. This returns us with a RDD which contain pairs, key = movieID and value = average rating.
   f. We apply mapreduce_join to obtain the corresponding movie names for the movie ids obtained in step e.
   g. We use takeordered function in pyspark to get 10 highest rated movies.

2. **For each genre, list the top state (which has given highest average rating).**
   Source Code File Name: **Question2.ipynb**
   Reason why MapReduce can’t be used:- We could not solve for all the genre in one map reduce. So we have applied iterative map reduce for each genre.
   The following pseudo code explains the series of map and reduce function usage in our code.
   a. Read users.csv, zipcodes.csv, rating.csv and movies.csv into collections.
   b. The basic idea is to replace userid column in rating.csv with the corresponding state of the user. For this, we create a userid to zipcode mapping and zipcode to city mapping, and then apply mapreduce_join operation to get userid to city mapping. We again apply the mapreduce_join operation to replace userid with state in the rating.csv
   c. For each genre:-
      1. Get the set of movies of this genre using map operation from movies.csv.
      2. We apply a join operation of the above set of movies with the result obtained in b.) to get the rating data *(state of the user rating the movie, user's rating)* only for these set of movies.
      3. We then apply a reduce operation to get average rating for each state.
      4. We use takeOrdered to get the state with maximum average rating.

3. **List the movies which have not been rated by anyone.**
   Source Code File Name: **Question3.ipynb**
   The following pseudo code explains the series of map and reduce function usages in our code.

a. Read rating.csv, and movies.csv into separate RDDs (df_rating, df_movies).
b. We apply a map operation to first convert the each string element in df_rating RDD to create a new RDD where each element is list of items.
c. We again apply a map operating to get a new RDD, with movieID as key, and 0 as value.
d. We apply map operations similar to b and c to df_rating as well, instead of 0 we take 1 as value.
e. We apply a union operation to merge the resultant RDDs of c and d.
f. We reduce the resultant RDD, using sum as our associative function for combining values for a key
g. Movies which were present in df_movies but were not present in df_rating will have value of 0. Rest of the movies will have value of 1.
h. We apply flatMap to get elements which have value of 0, or which were not rated by anyone
i. We apply mapreduce_join to obtain the corresponding movie names for the movie ids obtained in step e.

4. List all the user ids who rated only children movies between user age group 10 and 18.

Source Code File Name: **Question4.ipynb**

The following pseudo code explains the series of map and reduce function usages in our code.

a. Read users, movies and rating into RDDs.
b. Apply map operations on rf_users to convert string into list and then apply flatMap to get only users with age between 10 and 18.
c. Apply map on the above result to get a new RDD of pairs, with key as userID, and value a sentinel value ([‘epsilon’]).
d. We apply a series of map operations on the rating.csv data to get a RDD, with key as userID and value as movieID
e. We take union of RDDs we got in c. and d.
f. We apply a reduceByKey, merging the values in a list.
g. We apply a flatMap operation to get only those key-value pairs, in which value-list had ‘epsilon’ (sentinel value) and then apply a flatMap to get movieID, userID pairs. This way we have taken only key-value pairs from d. in which user age was between 10 and 18.
h. We apply operation similar to (g) to filter out pairs from output of (g) such that movie belonged to the children genre.
i. We use map to get list of userIDs from these pairs.

5. **Give number of users participated in rating movies city-wise.**
   Source Code File Name: **Question5.ipynb**
   The following pseudo code explains the series of map and reduce function usages in our code.
   a. Read users.csv, zipcode.csv, rating.csv into RDD/collections.
   b. Applying map operation on the collection of users.csv to get collection of pairs (key as userid, value as her\his zipcode.)
   c. We apply map operation and a reduce operation on rating.csv to get list of users who voted. Key - userid, value - 1
   d. We take mapreduce join of the two RDDs from b) and c) and apply a map operation to get only those pairs from b) where userid also belonged to c)
   e. Map operation is applied to the output. Key-zipcode, value 1
   f. Reduce operation is applied to the output, Key-zipcode, value-count of users who rated from that zipcode
   g. Map operation is applied on the zipcode.csv data to get mapping from zipcode to city name. Key-zipcode, value-city
   h. We use join on output of f) and g) and then apply map operation to get a RDD, in which key-city, value-count of users who rated in one of the zipcodes in the city
   i. Reduce operation on the output gives RDD in which key-city name and value-count of users who rated and from that city.

References:-

1. https://spark.apache.org/docs/0.9.0/index.html (retrieved 28/09/17)
2. https://spark.apache.org/docs/0.9.0/scala-programming-guide.html (retrieved 28/09/17)
3. https://spark.apache.org/docs/0.9.0/api/pyspark/index.html (retrieved 28/09/17)