**CSE 587 : ASSIGNMENT 2**

**TEAM MEMEBERS:**

1. SWATHI RAVINDRAN (50314300)
2. BALAGOPAL MADHUKUMAR

(50320449)

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a map procedure, which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a reduce method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. The model is a specialization of the split-apply-combine strategy for data analysis.

MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to minimize communication overhead.

A MapReduce framework (or system) is usually composed of three operations (or steps):

* Map: each worker node applies the map function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of the redundant input data is processed.
* Shuffle: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node.
* Reduce: worker nodes now process each group of output data, per key, in parallel.

1. **WORDCOUNT**:

Word count is the number of times a word has occurred in a given input. We can implement this by using MapReduce, which will have two parts, a mapper and a reducer module. Now let us look in detail into each of these components:

* **Mapper:**

1. Convert the line into lowercase.
2. Strip newlines from the line.
3. Replace non-alphanumeric characters from the line with blank space.
4. Split the lines into individual words and store them in an array.
5. Print word \t 1 for all words in the array. This will be the output from the mapper.
6. Clear the array and repeat for all lines.

* **Reducer**:

1. This is where the actual counting of the words happen.
2. Accepts input from the mapper result.
3. Checks if each word is equal to the previous list of already processed words from the mapper output.
4. If already present, increase the count of the word by 1 from the previous value.
5. Print all the words and its counts.
6. **TRIGRAM:**

A trigram is a group of three consecutively occurring words. We need to find the top 10 most occurring trigrams. We will implement this using the MapReduce which would again have us use the two modules- mapper and reducer. We will use two mappers and reducers for this.

* **Mapper 1**:

1. Convert the line into lowercase.
2. Strip the newlines from the lines.
3. Replace non-alphanumeric characters from the line with blank space.
4. Split the line into individual words and store them into a data array.
5. Add all these words into the final array and then clear the data array and repeat all the above steps.
6. Store the keys in an array.

* **Reducer 1:**

1. Reads each line from the output of the mapper.
2. Remove the newline and split the line into two words, one is the word and the other is the count.
3. Typecast the count to int.
4. If the word is equal to the previous word then add the count of the word and the previous count of that word.
5. If it is not, then add the current word to a list and set word and the count as current words.
6. Add the last word to the list.
7. Sort the words based on the count and then print the first 10.

* **Mapper 2:**

1. For each line of the input, print the line and this will be the input to the reducer 2.

* **Reducer 2**:

1. Performs the same function as the reducer 1.

**3. INVERTED INDEX:**

We will be mapping each word present in any of the three input files to the location that is the name of the file/files in which the word is constituted. This is again done using the MapReduce which would require two modules the mapper and the reduce.

The output will be stored in three text files.

* **Mapper :**

1. Imports the list of English stopwords from the corpus file of the ntlk package.
2. The input is always one line at a time.
3. The OS module from the standard utility library of python is used to access a function in it to interact with the operating system.
4. The ‘map\_input\_file’ is passed as a key to the os.getenv() which returns the value of the environment variable key and is stored in a string File\_Path.
5. The ‘File\_path’ string is split to retrieve only the name of the text file from the entire path entered.
6. The name is then converted to lowercase.
7. The data is retrieved from the line using the regular expression to get only the alphanumeric words and split and stored as a list
8. Stemming is performed using the Porter stemmer.
9. The stemmed words is checked for stopwords by comparing with a set of predefined stopwords.

* **Reducer :**

1. Creates a dictionary
2. Takes the output of the mapper as the input.
3. Each input is a line consisting of the word and its filename separated by a tab. The reducer code is iterated through the lines of input.
4. The input is split based on tab and is stored in respective word and posting. The word is stored as the key and the posting as the value.
5. The dictionary now contains all the words/terms and the names of all the text files in which the word is present.
6. Iterate through each key in the dictionary and then iterate through each filename for the key and then append it to a list, ie, postings.
7. Output or print the word and list of corresponding postings.

**4. RELATIONAL JOIN**:

We have to implement relational join for the given two input tables. The first table has Employee Id and Name. The second table has Employee Id, Salary, country and passcode. The relational join of these two tables should have Employee Id, Name, Salary, country and passcode. The primary key we use to do the relational join is Employee Id.

Again we use the MapReduce to implement this and so again we use the two modules mapper and reducer.

* **Mapper**:

1. Processes one line at a time from either of the files.
2. Distinguishes between the two files based on the number of columns each of the files have.
3. The output of the mapper is a key-value pair. The key is the primary id, which in this case is the EmployeeId. The rest of the columns are passed as the value.
4. The array is converted to a string format separated by commas and then passed as the output.

* **Reducer**:

1. Output of the mapper is received as the input to the receiver.
2. Every output from the mapper with the same key is passed to one reducer.
3. The first element indicates the key and the rest are the values.
4. The two tables are differentiated based on the length of the array storing the values corresponding to each key. If the length is more than 1, then the input is from the second table and stored into a dictionary else it is from the first table and stored into another dictionary.
5. The two dictionaries are iterated over the keys and employee id, name, salary, country and passcode are taken from the list and send as the output.

**5. KNN :**

KNN is one of the supervised machine learning algorithms used to solve both classification and regression problems. WE again implement KNN using MapReduce. We normalize the data before passing it through to the Hadoop environment.

* **Mapper:**

1. Each row is passed one at a time.
2. The input line is then split by commaa and then stored in an array.
3. The last element of the array is the label y and the rest of the element form the training data.
4. In mapper, the test data is read and the distance between each training and test data is calculated.
5. Finally, test data, distance, y is sent as the output to the reducer.
6. Here, test data is the key and the distance is the value.
7. The data is then converted to string and passed as output.

* **Reducer:**

1. The reducer takes the output of the mapper as the input.
2. The first element obtained is the key which is the test data.
3. Then the test data is split by comma and converted into a tuple. The tuple is then used as a key in a dictionary and the value for the dictionary is in the form of a list of lists.
4. For each row of the test data, the distance between it and each training data is calculated and the label of the training data and the distance is passed to the reducer.
5. For each test row, the list is sorted based on distances.
6. Top K labels are picked and stored in a list.
7. Finbally, the test row and its predicted labels are sent as output from the reducer.