Assignment No 5

Aim: To implement Map Reduce program for word count.

Objective:

Learn about MapReduce framework in Hadoop.
Implement MapReduce program for word count in R.

Theory:

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. **Map** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, **reduce** task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job. The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

MapReduce majorly has the following three Classes. They are:

Mapper Class
The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.

- Input Split
  It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
- Record Reader
  It interacts with the Input split and converts the obtained data in the form of Key- Value Pairs.

Reducer Class
The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.
Driver Class

The major component in a MapReduce job is a Driver Class. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

Advantages of MapReduce:

1. Parallel Processing:
In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

2. Data Locality:
Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- The master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

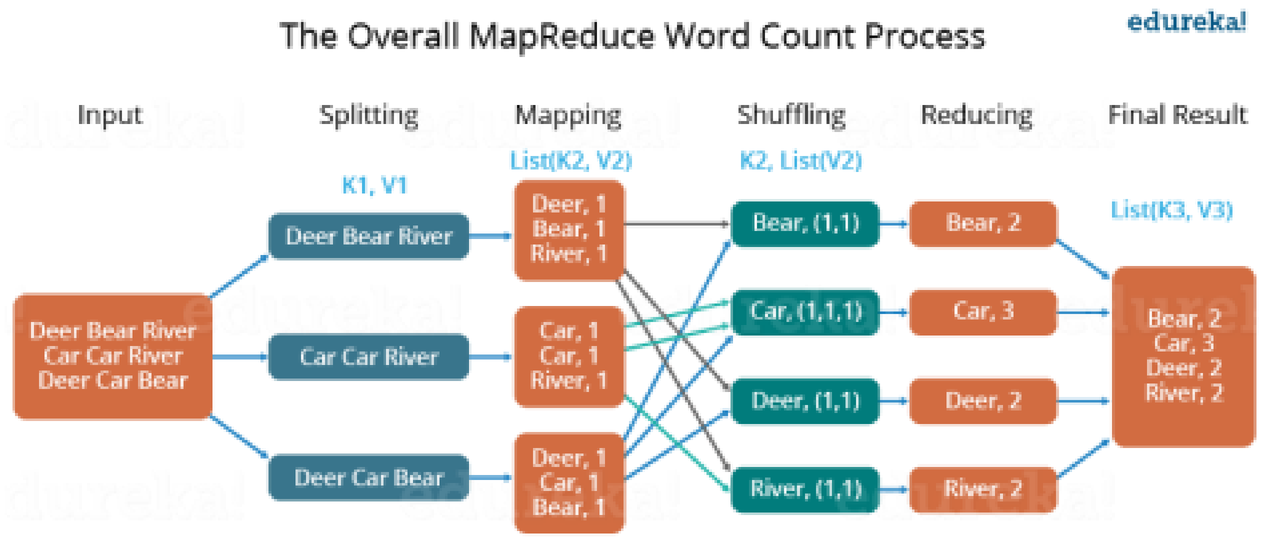
- It is very cost-effective to move processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

A Word Count Example of MapReduce:

Let us understand how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

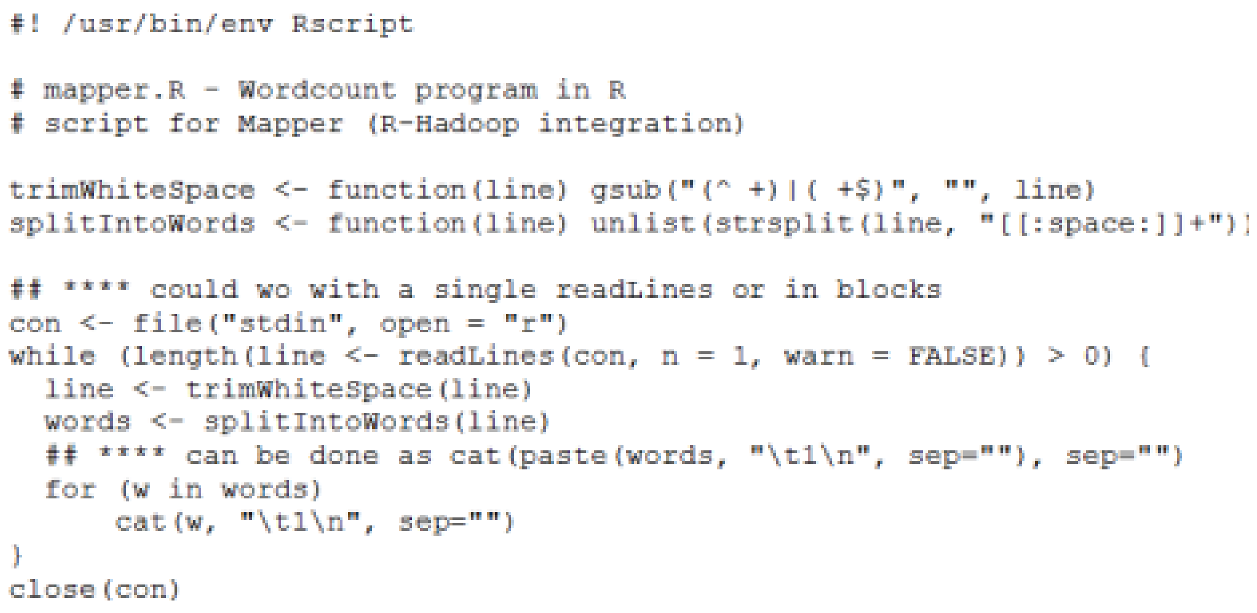
Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes. After
- the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc. Now,
- each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Code of Program:

mapper.R



```
#! /usr/bin/env Rscript

# mapper.R - Wordcount program in R
# script for Mapper (R-Hadoop integration)

trimWhiteSpace <- function(line) gsub("(^ +)|( +$)", "", line)
splitIntoWords <- function(line) unlist(strsplit(line, "[[:space:]]+")

## **** could wo with a single readLines or in blocks
con <- file("stdin", open = "r")
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  line <- trimWhiteSpace(line)
  words <- splitIntoWords(line)
  ## **** can be done as cat(paste(words, "\t1\n", sep=""), sep="")
  for (w in words)
      cat(w, "\t1\n", sep="")
}
close(con)
```

reducer.R

Output:

Conclusion:

In this assignment, we have successfully implemented Map Reduce program for word count.