BIG DATA ANALYTICS

ASSIGNMENT 4: Learning MapReduce Programming

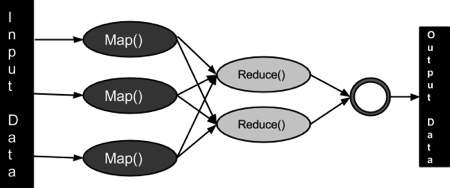
(Amitabh Saini, TY-E-4, 141541)

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.

**The Algorithm**

* Generally MapReduce paradigm is based on sending the computer to where the data resides!
* MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  + **Map stage**: The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  + **Reduce stage**: This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
* During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
* The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
* Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
* After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Example:

Input File Split Map Phase Shuffle & Sort Reduce Phase

This, 1

This, 1

This, 1

This, 1

is, 1

VIT, 1

This is VIT

This, 4

is, 4

VIT, 4

This, 1

is, 1

VIT, 1

This is VIT

This is VIT

This is VIT

is, 1

is, 1

is, 1

This is VIT

VIT, 1

VIT, 1

VIT, 1

This, 1

is, 1

VIT, 1

This is VIT

* Consider the above diagram, suppose we have a input file containing text as shown above.
* This file is first split in different blocks of particular size.
* These blocks are then passed to the map phase, which processes the count for each word.
* The map output is then shuffled and sort.
* The above output is then passed to the Reduce phase, which computes the output as above.
* LEARNING HASHMAP

Eg. Assume a file containing the website names unordered, find the no. of occurrence of each website.

import java.util.\*; //Contains HashMap

import java.io.\*; //Contains File Operations,BufferedReader

public class bda {

public static void main(String args[]) throws IOException {

BufferedReader br=null;

Map<String,Integer> countbywords = new HashMap<String,Integer>();

*//Creating a Hasmap of key as String and values as integer*

try {

br = new BufferedReader(new FileReader("C:\\Users\\user\\Desktop\\output.txt"));

String str;

while((str=br.readLine())!=null) *//Reading file line by line*

{

String next[]=str.split(",");

*//Splitting website and no. of visits*

int a = Integer.parseInt(next[1]);

*//Getting no. of visits given in line*

if(countbywords.containsKey(next[0]))

*//checking if website is already a key in hashmap*

{

countbywords.put(next[0],countbywords.get(next[0])+a) *//Incrementing count*

}

else

{

countbywords.put(next[0], a);

*//Creating a new key,value pair in HashMap*

}

}

System.out.println(countbywords.keySet());

*//Printing all keys in Hashmap*

System.out.println(countbywords.values());

*//Printing all values*

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

INPUT:

www.facebook.com,1

www.google.com,2

www.yahoo.com,3

www.facebook.com,1

www.facebook.com,2

www.facebook.com,3

www.yahoo.com,2

www.facebook.com,5

www.google.com,3

www.facebook.com,1

OUTPUT:

[www.yahoo.com, www.facebook.com, www.google.com]

[5, 13, 5]