# 301.1.6-Basics of MapReduce

**Map Reduce is not new**

* Finally, if we want to make something or achieve something, then you need not to do it in one go, you can divide the whole problem into different pieces i.e., initial data is the raw data and then you get the intermediate output and then we do the reduce.
* Thus, the idea was there earlier, now with the new network programming and all the network computing, we can achieve MapReduce distributed computing.
* To handle big data, we need to write MapReduce programs, we can’t write simple programs.

### MapReduce programs

* The conventional program to count the number records in a data file:

count=count+1

* The MapReduce program to count the number of records in a bigdata file:

count=count+1

cum\_sum=cum\_sum+sum

### More than just a MapReduce program

* The MapReduce program to count the number of records in a bigdata file:

count=count+1

cum\_sum=cum\_sum+sum

* Who will setup the network of machines and store the data locally?
* Who will divide and send the map program to local machines and call the reduce program on top of map?
* What if one machine is very slow in the cluster?
* What if there is a hardware failure in one of the machines?
  + It is not just MapReduce, it is not that simple. It is much more than MapReduce.

### Additional scripts for work distribution

1. We need to first setup a **cluster of machines**, then divide the whole data set into blocks and store them in local machines.
2. We also need to assign a **master node** that takes charge of all meta data, which block of data is on what machine.
3. We need to write a script that will take care of work scheduling, distribution of tasks and **job orchestration**.
4. We also need to assign **worker slots** to execute map and reduce functions.

### Additional scripts for efficiently

* We need to write scripts for load balancing (What if one machine is very slow in the cluster?).
* We also need to write scripts for data backup, replication and Fault Tolerance (What if the intermediate data is partially read).
* Finally write the map reduce code that solves our problem.

### Implementation of MapReduce is difficult

* Analysis on Bigdata can give us awesome insights.
* But, datasets are huge, complex and difficult to process.
* The solution is distributed computing or MapReduce.
* But looks like this data storage & parallel processing, job orchestration and network setup is complicated.
* What is the solution?
* Is there a readymade tool? Or platform that can take care of all these tasks.
  + Hadoop

# 301.1.5-MapReduce and Network Programming

* **Dividing The Overall Problem into Smaller Pieces**
  + Let us suppose we just want to simply find the number of lines or let’s say the data is from facebook, in a particular day how many likes are generated in facebook, that is what we want.
  + Thus we have the huge data set of all the ‘likes’, which will be in TB’s or PB’s, we divide the data into smaller pieces, let’s say each row in that data set represents one ‘like’ or each row represents one activity i.e., if we want to count the number of activity we can divide the whole data into smaller pieces, put them on all these lower end computers, now overall problem is, we want to count the number of activities.
  + We can divide this overall object also into smaller pieces. So we first count the activities in computer number one, whatever is the data on split one or data chunk 1, we can calculate the number of rows or we can calculate the number of activities, we can individually calculate the number of activities on all of these systems and we can do that parallely also and then once all these systems have locally calculated the number of rows or the number of activities, we can simply add them up later to get the final answer.
  + Now, we divide data and store them locally and then on each of the data, we run the task and this is called map.
  + Thus, on local systems at the map level or at very low level, we are calculating something, that is called map.
* Now, once we get output of the map then finally collate the results from local machines.
  + Let us take a simple three Node Cluster, 3 nodes – three computers, so we do a whole dataset into computer 1 , 2 and 3.
  + Then we take data chunk 1, 2 and 3 and the assign then to the 3 computers to work in parallel to calculate the number of row, andthis is called map.
  + Once we have the result of all these maps, then we can calculate REDUCE that is nothing but the sum of all these rows.
  + Thus, this is nothing but the distributed computing model.
* We can process bigdata in a parallel programming/distributed computing model.
* This is also known as **MapReduce** programming.

### Map Reduce and Network Programming

#### MapReduce: Programming Model

* MapReduce:
  + Map(the local/low level computation)
  + We go to each and every data set and data chunk wherever it is and calculate something. The output of the map is the input of the reduce.
  + Reduce(the collation of map results)
  + Reduce will take this output, calculate something over it and then finally give you the result.
* Thus we will be processing the data using special map() and reduce() functions.

**Map():** – The map() function is called on every item in the input and emits a series of intermediate key/value pairs(Local calculation). – All values associated with a given key are grouped together.

**Reduce():** – The reduce() function is called on every unique key, and its value list, and emits a value that is added to the output(final organization).

* Since, we can’t handle the huge data set or big data using normal computers or conventional tools, we are making use of distributed computing.
* You distribute the data first and then you split the overall problem in such a way that you can code it in a MapReduce format, which is called the distributed computing.

**301.2.3-Map Reduce Code for line count**

**LAB: Map Reduce for Line Count**

* Dataset: Stack\_Overflow\_Tags/final\_stack\_data.zip
* Unzip the file and see the size of the data.
* The dataset contains some stack overflow questions. The goal is to find out the total number of questions in the file(number of rows)
* Move the data to hdfs.
* Write a word count program to count the frequency of each word
* Take the final output inside a text file

**Solution**

Is the Hadoop started?

jps

Start Hadoop if not started already

start-all.sh

you can also use start-dfs.sh

Is the Hadoop started now? jps

check your files on hdfs

hadoop fs -ls /

Dataset is /Stack\_Overflow\_Tags/final\_stack\_data.zip. Unzip the data first. Unzipping takes some time

sudo unzip /home/hduser/datasets/Stack\_Overflow\_Tags/final\_stack\_data.zip -d /home/hduser/datasets/Stack\_Overflow\_Tags/

Bring the data onto hadoop HDFS. The file size is huge. This step takes some-time

hadoop fs -copyFromLocal /home/hduser/datasets/Stack\_Overflow\_Tags/final\_stack\_data.txt /stack\_overflow\_hdfs

Check the data file on HDFS

hadoop fs -ls /

check your current working directory

cd

Goto hadoop bin

cd /usr/local/hadoop/bin/

It is imporatant to make your PWD(present working directory) as $hadoop/bin

Open an editor with a file name **LineCount.java**

sudo gedit LineCount.java

Copy the below java code, paste in your file and save your file

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class LineCount{

public static class LineCntMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

Text keyEmit = new Text("Total Lines");

private final static IntWritable one = new IntWritable(1);

public void map(LongWritable key, Text value, Context context){

try {

context.write(keyEmit, one);

}

catch (IOException e) {

e.printStackTrace();

System.exit(0);

}

catch (InterruptedException e) {

e.printStackTrace();

System.exit(0);

}

}

}

public static class LineCntReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterable<IntWritable> values, Context context){

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

try {

context.write(key, new IntWritable(sum));

}

catch (IOException e) {

e.printStackTrace();

System.exit(0);

}

catch (InterruptedException e) {

e.printStackTrace();

System.exit(0);

}

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "line count2");

job.setJarByClass(LineCount.class);

job.setMapperClass(LineCntMapper.class);

job.setCombinerClass(LineCntReducer.class);

job.setReducerClass(LineCntReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

To compile this program, use the below command

hadoop com.sun.tools.javac.Main LineCount.java

Create the jar file which is named as **lc.jar**

jar cf lc.jar LineCount\*.class

Run linecount program, output will be automaically routed to

hadoop jar lc.jar LineCount /stack\_overflow\_hdfs /usr/stack\_overflow\_out

Check the output here <http://localhost:50070/explorer.html#/>

Have a look at the output

hadoop fs -cat /usr/stack\_overflow\_out/part-r-00000

We can take the output to a text file

hadoop fs -cat /usr/stack\_overflow\_out/part-r-00000 >> /home/hduser/Output/stack\_overflow\_out.txt

**301.2.4-Map Reduce Code for Finding Average**

**LAB: Map Reduce Function-Average**

* Dataset: Stock\_Price\_Data/stock\_price.txt
* The dataset contains stock price data collected for every minute.
* Find the average stock price.

**Solution**

Is the Hadoop started?

jps

Start Hadoop if not started already

start-all.sh

you can also use start-dfs.sh

Is the Hadoop started now? jps

check your files on hdfs

hadoop fs -ls /

Bring the data onto hadoop HDFS

hadoop fs -copyFromLocal /home/hduser/datasets/Stock\_Price\_Data/stock\_price.txt /stock\_price

Check the data file on HDFS

hadoop fs -ls /

check your current working directory

cd

Goto hadoop bin

cd /usr/local/hadoop/bin/

It is imporatant to make your PWD(present working directory) as $hadoop/bin

Open an editor with a file name **AvgMapred.java**

sudo gedit AvgMapred.java

Copy the below java code, paste in your file and save your file

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AvgMapred {

/\*

\* data schema(tab separated) : 2000.16325 4664654.78955 46513123.134165

\*/

public static class MapperClass extends

Mapper<LongWritable, Text, Text, FloatWritable> {

public void map(LongWritable key, Text empRecord, Context con)

throws IOException, InterruptedException {

String[] word = empRecord.toString().split("\\n");

String flg = " ";

try {

Float salary = Float.parseFloat(word[0]);

con.write(new Text(flg), new FloatWritable(salary));

} catch (Exception e) {

e.printStackTrace();

}

}

}

public static class ReducerClass extends

Reducer<Text, FloatWritable, Text, Text> {

public void reduce(Text key, Iterable<FloatWritable> valueList,

Context con) throws IOException, InterruptedException {

try {

Double total = (double) 0;

int count = 0;

for (FloatWritable var : valueList) {

total += var.get();

//System.out.println("reducer " + var.get());

count++;

}

Double avg = (double) total / count;

String out = "Total: " + total + " :: " + "Average: " + avg;

con.write(key, new Text(out));

} catch (Exception e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

Configuration conf = new Configuration();

try {

Job job = Job.getInstance(conf, "FindAverageAndTotalSalary");

job.setJarByClass(AvgMapred.class);

job.setMapperClass(MapperClass.class);

job.setReducerClass(ReducerClass.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

} catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

To compile this program, use the below command

hadoop com.sun.tools.javac.Main AvgMapred.java

Create the jar file which is named as **avg.jar**

jar cf avg.jar AvgMapred\*.class

Run average program, output will be automaically routed to

hadoop jar avg.jar AvgMapred /stock\_price /usr/stock\_price\_out1

Part of the output from above line in Terminal looks like below :

Have a look at the output

hadoop fs -cat /usr/stock\_price\_out1/part-r-00000

We can take the output to a text file

hadoop fs -cat /usr/stock\_price\_out1/part-r-00000 >> /home/hduser/Output/stock\_price\_out.txt