Experiment No.2

MapReduce Program on a Real Time Dataset

**Mohit Badve**

**BE COMPS  
2017130006**

**Aim:** To implement a MapReduce program on a real time dataset.

**Theory:**

**MapReduce:**

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

* The map task is done by means of Mapper Class
* The reduce task is done by means of Reducer Class.

Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.

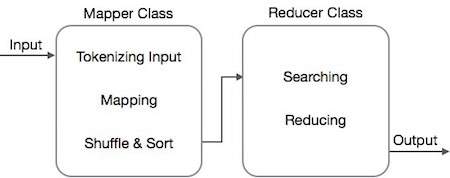


Fig.1 General MapReduce Approach

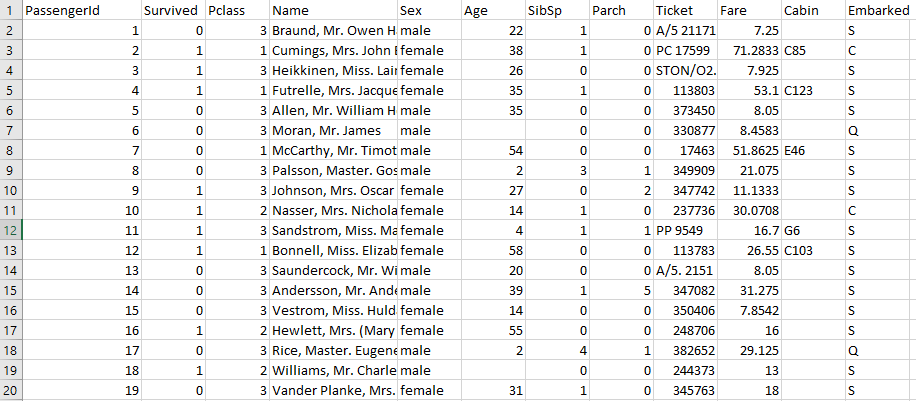
MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems.

These mathematical algorithms may include the following −

* Sorting
* Searching
* Indexing

**Dataset:**

The Titanic Dataset (subset).



**Problem Definition:**

Problem statement 2:

In this problem statement, we will find the number of people died or survived in each class with their genders and ages.

These values will be passed to the shuffle and sort phase and are further sent to the reducer phase where the aggregation of the values is performed.

**Implementation:**

**1. Setup Environment.**

Setup Hadoop MapReduce environment for java by adding Hadoop JAR files in your java project.

Common JARs

MapReduce JARs

Yarn JARs

HDFS JARs

**2. Creating a Mapper.**

public class Female\_survived {

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

private Text people = new Text();

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

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

String line = value.toString();

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

if(str.length>6){

String survived=str[1]+" "+str[4]+" "+str[5];

people.set(survived);

context.write(people, one);

}

}

}

**3. Creating a Reducer**

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

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

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

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

}

}

**Output:**

0 female 10 1

0 female 11 1

0 female 14 1

0 female 14.5 1

0 female 16 1

0 female 17 1

0 female 18 5

0 female 2 4

0 female 20 2

0 female 21 3

0 female 22 2

0 female 23 1

0 female 24 2

0 female 25 3

0 female 26 2

0 female 27 1

**Conclusion:**

The MapReduce approach is a highly scalable processing technique for distributed computing. Instead of moving the data towards the computation, the computations are brought to the data nodes which improves the overall performance.

In the above implementation the map function extracts Marital Status and Working hours from the input records and then “average” as a aggregation function is implemented in the reduce phase which gives us the mean working hours per week for different Marital Statuses.