

DS 503 - Project 1

By :

Abdulaziz Alajaji - [asalajaji@wpi.edu](mailto:asalajaji@wpi.edu)

Yousef Fadila - [yousef@fadila.net](mailto:yousef@fadila.net)

---

### Question 1 : Creating Datasets

The main class to create the Datasets is DataGeneratorMain.java.

The code is very short and straightforward

```
package edu.wpi.ds503;

public class DataGeneratorMain {

    public static void main(String[] args) {

        DataGenerator[] dataGenerators = new DataGenerator[]{ new CustomersGenerator(50000),
                                                                new TransactionsGenerator(5000000) };

        for (DataGenerator dataGenerator : dataGenerators) {
            dataGenerator.generate();
        }
    }
}
```

---

Both CustomersGenerator and TransactionsGenerator classes extend DataGenerator abstract class which has abstract method named: "generate()".

The Generator constructor receive number of records as an argument in order to allow creating small samples.

---

### Question 2- uploading Data into Hadoop

Uploading the data from local file system to HDFS was done using the put command:

```
root@hadoop-VirtualBox: /media/sf_DS503/github-big data managment
sbin/  selinux/ srv/    sys/
root@hadoop-VirtualBox:/usr/share/hadoop# cd /d
data/ dev/  dfs/
root@hadoop-VirtualBox:/usr/share/hadoop# cd /usr/
bin/  etc/    games/  include/ lib/    lib64/  libexec/ local/  native/ sbin/    share/  src/
root@hadoop-VirtualBox:/usr/share/hadoop# cd /media/
sf_DS503/
VBOXADDITIONS_5.1.14_112924/
root@hadoop-VirtualBox:/usr/share/hadoop# cd /media/sf_DS503/
root@hadoop-VirtualBox:/media/sf_DS503# ls
CC.ppt      Hadoop-Easy-MapReduce-master.zip  Hadoop.The.Definitive.Guide.4th.Edition.2015.3.pdf
github-big data managment          Hadoop-Pig.pptx                  Intro.pptx
HadoopAnalytics-1.pptx            Hadoop-Platform.pptx            mining of massive dataset.pdf
Hadoop-Easy-MapReduce-master       Hadoop-Streaming.pptx           passwords.txt
root@hadoop-VirtualBox:/media/sf_DS503# cd github-big\ data\ managment/
root@hadoop-VirtualBox:/media/sf_DS503/github-big data managment# ls
Customers  github-big data managment.inl  out  README.md  src  Transactions
root@hadoop-VirtualBox:/media/sf_DS503/github-big data managment# hadoop fs -put Transactions /user/hadoop/
Warning: $HADOOP_HOME is deprecated.

root@hadoop-VirtualBox:/media/sf_DS503/github-big data managment#
root@hadoop-VirtualBox:/media/sf_DS503/github-big data managment# hadoop fs -put Customers /user/hadoop/
Warning: $HADOOP_HOME is deprecated.

root@hadoop-VirtualBox:/media/sf_DS503/github-big data managment#
```

### 3.1) Query 1

Query 1 was implemented using map only job

The command to run the job:

```
hadoop jar hw1.jar edu.wpi.ds503.hadoop.Query1 /user/hadoop/Customers
/user/hadoop/query1
```

Screenshot of the output:

**File: [/user/hadoop/query1/part-r-00000](#)**

---

Goto :

---

[Go back to dir listing](#)

[Advanced view/download options](#)

---

[View Next chunk](#)

---

1	6
3	6
5	6
8	5
9	4
10	4
11	6
13	2
14	5
15	2
16	5
18	6
21	4
29	2
30	6
32	5
33	2
36	5
38	3

---

### 3.2) Query 2

Query2 was implemented with single map-reduce job with combiner.

The command to run the job:

***hadoop jar hw1.jar edu.wpi.ds503.hadoop.Query2 /user/hadoop/Transactions  
/user/hadoop/query2***

**Screenshot of the output:**

**File: [/user/hadoop/query2a/part-r-00000](#)**

---

Goto :

---

[Go back to dir listing](#)

[Advanced view/download options](#)

---

[View Next chunk](#)

---

```
1,103,50148.59
2,107,50261.043
3,98,50168.57
4,112,54094.03
5,115,61562.008
6,94,45313.234
7,102,52583.305
8,110,55984.586
9,104,53693.332
10,98,51934.914
11,97,51153.92
12,106,58410.676
13,92,45672.08
14,88,46433.1
15,92,41465.414
```

We define new writeable object “**CustomerQuery2Data**” to be passed between Map, Combiner and reducer.

```
CustomerQuery2Data {
    Count;
    Sum;
}
```

The Mapper will set count to 1 and sum to transTotal for each single transaction

The Combiner will set count and sum to the aggregated values for each customer.

The reducer has the same functionality like the Combiner except that it produces Text, not CustomerQuery2Data

---

### 3.3) Query 3

Query3 was implemented with 2 mappers with 1 reducer job.

The command to run the job:

```
hadoop jar hw1.jar edu.wpi.ds503.hadoop.Query3 /user/hadoop/Customers
/user/hadoop/Transactions /user/hadoop/query3
```

Screenshot of the output:

**File: [/user/hadoop/query3/part-r-00000](#)**

Goto :

[Go back to dir listing](#)

[Advanced view/download options](#)

[View Next chunk](#)

```
1, VYNsPKuLNG, 1874.283, 103, 50148.598, 1
2, BBZHwlbCax, 3971.3115, 107, 50261.043, 1
3, WvmOTNvyrgmHoBcAn, 4988.2686, 98, 50168.574, 1
4, SkPzbHNaCgCDAEvT, 8665.687, 112, 54094.03, 1
5, wwJIIsrUEKmbGCSafbYS, 4547.3994, 115, 61561.992, 1
6, DhpFSWrNpqkNTj, 5904.812, 94, 45313.223, 1
7, tGGHlWtHTOCRZNsJD, 7373.5923, 102, 52583.312, 1
8, XbFSzaWvDBfudLt, 2880.1619, 110, 55984.582, 1
9, YmtmyoJybyIwDRcaYb, 5673.941, 104, 53693.34, 1
10, cyUGxtfTomI, 6809.374, 98, 51934.906, 1
11, GDtofsWuyRZDMUIUn, 672.68494, 97, 51153.92, 1
12, vNkTWdbPvqFiHJAMuiY, 3658.2097, 106, 58410.684, 1
13, wLwBGvV0afYVpgsfv, 1126.6558, 92, 45672.086, 1
14, inenaKLEDWhJclhZV, 5710.987, 88, 46433.098, 1
```

We define new writeable object “**Query3Data**” to be passed between the 2 Mapers and the reducer.

```
Query3Data {
    name
    salary
    transTotal
    transItems
}
```

Customers Mapper will fill in name and salary and writes the object with CustomerId key.  
Transaction Mapper will fill in transTotal and transItems and writes the object with CustomerId key

In the reducer, the indicator whether the data comes from Transactions or Customers is **if name is defined or not!** There is no need to define another flag.

### 3.4) Query 4

Query4 was implemented with single map-reduce job.

The command to run the job:

```
hadoop jar hw1.jar edu.wpi.ds503.hadoop.Query4 /user/hadoop/Customers  
/user/hadoop/Transactions /user/hadoop/query4
```

Screenshot of the output:

**File: [/user/hadoop/query4a/part-r-00000](#)**

Goto :

[Go back to dir listing](#)

[Advanced view/download options](#)

```
1,4974,10.005665,999.9992  
2,5069,10.000118,999.99994  
3,4943,10.002479,999.9995  
4,4927,10.010149,999.99774  
5,5019,10.000885,999.99927  
6,5083,10.001594,999.99695  
7,4949,10.00059,999.99774  
8,5007,10.000177,999.9991  
9,5088,10.000944,999.99963  
10,4941,10.000708,999.9995  
|
```

In order to implement the query with single map-reducer job, motivated by the fact that the customers files is relatively small. We decide to build a lookup table in the setup that maps each customer to country code.



```

protected void setup(Context context) throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    String customerPath = conf.get(CUSTOMER_PATH);
    try {
        Path pt = new Path(customerPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(pt)));
        String line;
        line = br.readLine();
        while (line != null) {
            String[] fields = line.toString().split(",");
            int customerId=Integer.parseInt(fields[CUST_CUSTOMER_ID]);
            int countryCode=Integer.parseInt(fields[CUST_CONTRY]);
            custToCountryMap.put(customerId, countryCode);
            countryCustomers[countryCode - 1] = countryCustomers[countryCode - 1] + 1 ;
            line = br.readLine();
        }
    } catch (Exception e) {
        throw new IOException("UNEXPECTED,error",e);
    }
}

```

The disadvantage of this approach is that each mapper has to read the customer's file to build this lookup table. The advantage is that it saves another map-reducer job to do the join based on customer Id.

Based on the fact that customers file size is less than 2MB, we favor to pay the penalty of each mapper has to read this 2 mb file instead of doing double map-reduce job to do the join. Since we already read the file, line by line, and the number of countries is limited to 10. We choose to count customers per country during the setup as the cost of counting the customer is negligible in both space and time.(done in the same loop that make the look-up table)

---

### 3.5) Query 5

Query5 was implemented with 2 map-reduce jobs followed by map only job.

The command to run the job:

```

hadoop jar hw1.jar edu.wpi.ds503.hadoop.Query5 /user/hadoop/Customers
/user/hadoop/Transactions /user/hadoop/query5

```

Screenshot of the output:

## Contents of directory [/user/hadoop/query5](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time
<a href="#">final</a>	dir				2017-02-10 20:33
<a href="#">step1</a>	dir				2017-02-10 20:32
<a href="#">step2</a>	dir				2017-02-10 20:32

[Go back to DFS home](#)

### Final result screenshot

The output is the name, Trans number; we print the Trans number as validation to manually check if all reported customers will be higher than output from step2

**File: [/user/hadoop/query5/final/part-r-00000](#)**

Goto :

[Go back to dir listing](#)

[Advanced view/download options](#)

[View Next chunk](#)

```
VYNPKuLNG,103
BBZHwlbCax,107
SkPzbHNaCgCDAEvT,112
wvJIsrUEKmbGCSafbYS,115
tGGHlWtHTOCRZnsjD,102
XbFSzaWvDBfudLt,110
YmtmyoJybyIwDRcaYb,104
vNkTWdbPvqFiHJAMuiY,106
VjmTKtrozk,105
aqqpfImCwOTPBwddb,105
aZsmOuLVonLwRes,105
sbOCQHmCuqAwJyx,104
```

The first map-reduce job of this question is running query 2.



```

Job job = Job.getInstance(conf, "Project1, Query5 - step1");
job.setJarByClass(Query5.class);

// first step is running Query2.
job.setMapperClass(Query2.CustomersQuery2Mapper.class);
job.setCombinerClass(Query2.CustomersQuery2Combiner.class);
job.setReducerClass(Query2.CustomersQuery2Reducer.class);

job.setOutputKeyClass(NullWritable.class);
job.setOutputValueClass(Text.class);

job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(Query2.CustomerQuery2Data.class);
FileInputFormat.addInputPath(job, new Path(args[1]));
FileOutputFormat.setOutputPath(job, new Path(args[2] + STEP1_OUTPUT));

```

That because in query 2 we count the transaction per customer.

The 2nd step is calculating the average; done by map-combiner-reducer job. We define new writeable object:

```

Query5Data {
    Count;
    Average;
}

```

The mapper will go over the result from query 2. And fill in this object with count =1 ; average=total trans

```

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String[] fields = value.toString().split(",");
    // 19,105,55984.484 (query2 output is CustomerId, TotalTrans,Sum
    float transNum=Float.parseFloat(fields[TRANS_NUM]);
    FloatWritable transNumFloatWritable = new FloatWritable(transNum);
    context.write(NullWritable.get(), new Step2Query5Data().set(one, transNumFloatWritable));
}

```

The combiner will aggregate the count and calculate the new average.

```

public static class Step2Query5Combiner
    extends Reducer<NullWritable, Step2Query5Data, NullWritable, Step2Query5Data> {

    public void reduce(NullWritable key, Iterable<Step2Query5Data> values, Context context)
        throws IOException, InterruptedException {
        float sum = 0;
        int count = 0;

        for (Step2Query5Data val : values) {
            float average = val.getAverage().get();
            int _count = val.getCount().get();

            sum += average * _count;
            count += val.getCount().get();
        }

        float newAverage = sum/count;
        context.write(key, new Step2Query5Data().set(new IntWritable(count),
            new FloatWritable(newAverage)));
    }
}

```

And finally, the reducer will calculate the global average

```

public static class Step2Query5Reducer
    extends Reducer<NullWritable, Step2Query5Data, NullWritable, Text> {

    public void reduce(NullWritable key, Iterable<Step2Query5Data> values, Context context)
        throws IOException, InterruptedException {
        float sum = 0;
        int count = 0;

        for (Step2Query5Data val : values) {
            float average = val.getAverage().get();
            int _count = val.getCount().get();

            sum += average * _count;
            count += val.getCount().get();
        }

        float newAverage = sum/count;
        // print count for validation
        Text result = new Text(count + "," + newAverage);
        context.write(NullWritable.get(), result);
    }
}

```

The 3rd step is map only task.

In the setup of the map we create a lookup table to map customerId to customer name.  
And we read the global average from step 2

```

protected void setup(Context context) throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    String customerPath = conf.get(CUSTOMER_PATH);
    String step2OutputPath = conf.get(STEP2_PATH);
    try {
        Path pt = new Path(customerPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(pt)));
        String line;
        line = br.readLine();
        while (line != null) {
            String[] fields = line.toString().split(",");
            int customerId=Integer.parseInt(fields[CUST_CUSTOMER_ID]);
            custIdToNameMap.put(customerId, fields[CUST_NAME]);
            line = br.readLine();
        }
        // read the one line of the putput of step 2. format COUNT,AVERAGE.
        pt= new Path(step2OutputPath);
        br = new BufferedReader(new InputStreamReader(fs.open(pt)));
        line = br.readLine();
        String[] fields = line.toString().split(",");
        global_average = Float.parseFloat(fields[1]);
    } catch (Exception e) {
        throw new IOException("UNEXPECTED,error",e);
    }
}

```

The mapper of this step is very simple, it goes over the step1 (query 2 result) and filter out all customers have less or equal to the global average

```

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    String[] fields = value.toString().split(",");

    // 19,105,55984.484 (query2 output is CustomerId, TotalTrans,Sum
    int customerId=Integer.parseInt(fields[CUST_CUSTOMER_ID]);
    int transNum=Integer.parseInt(fields[TRANS_NUM]);

    if (transNum > global_average) {
        String name = custIdToNameMap.get(customerId);
        if (name == null) {
            throw new IOException("UNEXPECTED, customer is not fount");
        }
        // print the transNum to validate that all of customers will be higher than output from step2
        context.write(NullWritable.get(), new Text(name + "," + transNum));
    }
}

```