Map/Reduce Implementation

Submitted by: Team 15 - Deepak Thipeswamy (1001235195)

Under Supervision of: Dr. Sharma Chakravarthy and GTA, Mr. Jay Bodra

Department of Computer Science,

University of Texas at Arlington

**Acknowledgements**

My sincere thanks to Dr. Sharma Chakravarthy who gave us brilliant lectures on Map/Reduce before the project and throughout and without his guidance, implementing the project would have been really difficult.

Deepak Thipeswamy (1001235195)

**Overview**

The project involves implementation of a simple map/reduce paradigm using which students have to analyze the weather datasets of the cities given and generate <Key, Value> pairs indication the Station Ids having the similar weather. The project scope is to implement 2 Map/Reduce jobs each of which is used to arrive at the desired result.

MapReduce has 2 main parts:

• Map: processes a key value pair to generate a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. • Reduce: accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values.

1. Mapper 1 – Key should be <Station Id + Month + Section>

Value should be <Temperature + Dew Point + Wind Speed>

1. Reducer 1 – Generates averages of Temperature, Dew Point and Wind Speed for each of the keys presented from the Mapper 1 and feeds it to Mapper 2.
2. Mapper 2 – Takes input from Reducer 1 and generates Key <Station Id + Month> and Value as < Section wise averages>
3. Reducer 4 – Generates the final output by displaying the 12 attribute vector having the averages of Temperature, Dew Point and Wind Speed indicating them section wise for each of the Station Id for a given month.

**Overall Status**

The Project has been **successfully completed**. Two mappers and two reducers are being implemented.

* Successful installation and setup of Hadoop 2.6.0.
* Successful implementation of Mappers and Reducers and job chaining
* Successful deployment and testing.

**Configuration**:

1. Hadoop 2.6.0
2. **OS** : Ubuntu 14.04 (Running in a VM)
3. **Java** : Java 1.8.0\_66

**Implementation Details:**

**Setup:**

* After going through the project material, I got a fair idea about the setup so I started looking for other documents or steps for installation. The used the below link for the following installation

<http://tecadmin.net/setup-hadoop-2-4-single-node-cluster-on-linux/>

* I made sure I made all the right configurations for the same and I was able to run a simple Word Count Program provided.
* **Start the Nodes:**

*start-dfs.sh*

*start-yarn.sh*

**HDFS file system:**

* HDFS file system is what Hadoop uses for the mechanism it provides for map/reduce.
* The input file has to be loaded into the HDFS file system so that Hadoop is able to understand the input and carry on with its process.
* The input gets uploaded as block of 64MB and Hadoop uses it for input splitting and assigning to mappers.
* The input can be loaded to the HDFS filesystem using :

*hadoop dfs -copyFromLocal input inputhd*

**Compile and generate jar:**

*hadoop@ubuntu:~/hadoop$ bin/hadoop com.sun.tools.javac.Main Weather.java*

*hadoop@ubuntu:~/hadoop$ jar cf wc.jar Weather\*.class*

**Run Map/Reduce:**

*hadoop@ubuntu:~/hadoop$ hadoop jar wc.jar Weather inputhd outputhd*

**Copy output from hdfs to local filesystem**

*hadoop/bin/hadoop dfs -copyToLocal /user/hadoop/output finalOutput*

**Display output on screen**

*hadoop dfs -cat output/part-r-00000*

**Remove output directory (depends on where is the output directory is)**

*hadoop dfs -rmr /user/hduser/output*

*hadoop dfs -rmr /user/hduser/tempoutput*

**Implementation of Mappers and Reducers:** The implementation of the mapper is very simple, the decision of key, value pairs drives the implementation.

The data for each hour starts from 0, so, 5:01am data will be found after 5th hour i.e., 5 in dataset given.

**5:01 am to 11:00 am falls in 5,6,7,8,9,10**

**11:01 am to 5:00 pm falls in 11,12,13,14,15,16**

**5:01 pm to 11:00 pm falls in 17,18,19,20,21,22**

**11:01 pm to 5:00 am falls in 23,0,1,2,3,4**

**So, 5-10 = S1 and 11-17 = S2 and 18-22 = S3 and 23 and 0-4 = S4**

Mapper 1 – Based on the input, Station id + month + Section (depending on hours) is chosen as key and the corresponding row of data is written into the context so that for this key, there will be that many values.

Reducer 1 – Receives the key and gets key value pairs and generates the average of Temperature, Dew Point and Wind speed and the job chaining happens.

Mapper 2 – Reads the output written by Reducer 1 and generates Key value pairs based on Station Id and Month. The values are Section wise averages of the data.

Reducer 2 – Receives input from the Mapper 2 and processes results and creates a 12 attribute vector based on Station Id and month and displays Section wise averages into 4 parts.

**Sample Output: 2007 year’s dataset (Did not round it to keep the data more accurate)**

StId + month 12 attribute vector of averages of data section wise (by pipe).

690190\_01 S1\_44.22486338797815,25.519672131147523,10.254098360655728|S2\_45.66633879781421,25.519672131147583,10.254098360655721|S3\_45.12767759562836,25.519672131147598,10.254098360655757|S4\_44.63035519125683,25.51967213114757,10.254098360655732

690190\_02 S1\_49.09903333333333,29.23000000000001,8.488000000000001|S2\_50.638733333333356,29.22999999999995,8.488000000000007|S3\_49.38023333333333,29.230000000000008,8.488000000000005|S4\_50.41423333333331,29.230000000000008,8.487999999999985

690190\_03 S1\_65.56187500000001,48.19166666666665,8.906250000000005|S2\_65.66559027777781,48.19166666666675,8.906249999999996|S3\_65.97968749999995,48.191666666666734,8.906249999999993|S4\_66.13739583333333,48.19166666666669,8.906249999999996

690190\_04 S1\_65.75482638888894,50.470833333333395,9.547916666666671|S2\_66.43826388888895,50.47083333333338,9.547916666666689|S3\_66.06743055555555,50.47083333333339,9.547916666666675|S4\_65.89802083333328,50.47083333333338,9.547916666666671

690190\_05 S1\_106.88900641025637,62.57884615384616,6.805769230769243|S2\_107.76192307692308,62.57884615384615,6.805769230769218|S3\_107.57618589743596,62.57884615384624,6.805769230769242|S4\_107.63,62.57884615384622,6.8057692307692355

690190\_06 S1\_78.56152173913048,66.10434782608708,6.941304347826078|S2\_78.65539855072466,66.104347826087,6.94130434782608|S3\_79.15536231884059,66.10434782608692,6.941304347826081|S4\_78.4652536231884,66.10434782608705,6.941304347826077

690190\_07 S1\_79.39444444444446,68.04583333333338,5.1479166666666645|S2\_79.72715277777777,68.04583333333329,5.147916666666672|S3\_79.87246527777778,68.04583333333339,5.147916666666664|S4\_79.21899305555557,68.04583333333335,5.147916666666664

690190\_08 S1\_83.60087719298237,68.08596491228079,6.354385964912271|S2\_83.5553216374269,68.08596491228066,6.354385964912265|S3\_83.99213450292397,68.0859649122807,6.354385964912286|S4\_83.79912280701753,68.08596491228079,6.3543859649122725

690190\_09 S1\_78.47692028985503,65.58260869565207,6.391304347826087|S2\_78.98086956521738,65.5826086956522,6.391304347826081|S3\_78.87496376811593,65.58260869565216,6.391304347826075|S4\_79.21090579710147,65.5826086956522,6.391304347826086

690190\_10 S1\_68.00123333333335,46.02400000000005,8.887999999999993|S2\_68.8081,46.02399999999997,8.88799999999999|S3\_69.07843333333328,46.02399999999999,8.88800000000001|S4\_68.11216666666665,46.023999999999994,8.888

690190\_11 S1\_58.252166666666724,40.263333333333364,8.786666666666667|S2\_58.968388888888875,40.26333333333331,8.786666666666683|S3\_57.528083333333335,40.26333333333332,8.786666666666665|S4\_58.20124999999997,40.263333333333314,8.78666666666667

690190\_12 S1\_46.57529569892472,30.962903225806436,8.633870967741942|S2\_47.379408602150555,30.962903225806517,8.633870967741935|S3\_47.679569892473125,30.962903225806496,8.633870967741927|S4\_47.8382795698925,30.962903225806517,8.633870967741938

**Performance Measure:**

*Hadoop 2.6.0 does not have a JobTracker associated with it. Resource Manager does not write the comprehensive log for the same. Hence omitting the analysis results as discussed.*

By using the following API, we can configure the number of map and reduce tasks for the jobs.

**FileInputFormat.setMaxInputSplitSize(job1, 67108864); -** The number of splits will decide the number of mappers so, in Hadoop 2.6.0 if we specify the chunk size lesser or greater than the standard, we will be able to change the number of map tasks.

//33554432 - 32mb

//67108864 - 64mb

//134217728 - 128mb

**job1.setNumReduceTasks(7); -** This API is used to set the number of reducers that can be used to set the number of reducers per job.

**Division of Labor**

The project was completed single handedly as is a one student team.

The project setup took 2 – 3 hrs

The coding for the project took 5-6 hrs

The performance analysis took 5-6 hrs

Overall time consumed for this project was ~15 hrs

**Logical Errors**

* **Input Datatype mismatch** – I encountered this problem where I had designed with different data types of inputs for each mapper and reducer. I was using **Text** for the first mapper and when the second Mapper was getting invoked, there was a datatype mismatch. Hence I went ahead with Text as the datatype for both map/reduce jobs.
* **API Compatibility –** To test for different configuration, I had to change the number of map tasks and the reduce tasks for the analysis. The API I found out by searching online was **conf.setNumOfMaps(<integer>)** but I had not considered the possibility of a deprecation of the same API. Then I found out that for Hadoop 2.6.0 is -

**FileInputFormat.setMaxInputSplitSize(<job>, <size in bytes>);**

* This API takes does not explicitly reduce the number of mapper tasks but, an understanding of the input splits made me realize that if I’m able to force the number of splits to increase, then I can accomplish forcing the system to use those many mapper tasks (Every split is assigned a mapper)

**job.setNumReduceTasks(<number of tasks>)**

* For increasing the number of tasks, this API can be called and it explicitly dictates the number of reducer tasks it should assign for the current job.
* **Key, Value Pairing –** It took some time to realize that for any given station id, we should not consider the average temperature or dew point or wind speed for any given year, but we should consider for any given station id and month. So there will be a 12 attribute vector generated for each month for a given station id.
* **String Manipulation –** Even though these are not logical errors, I wanted to mention this because there is a lot of String manipulation by reading the input from the dataset and generating key value pairs. A small mistake in string manipulation, the data sets will be mapped and reduced wrongly.
* **Hour’s confusion –** The output data was wrong because the section wise division was strictly dependent on the time of the day. 5:01 am falls into 5th hour of the data set. Considering the dataset starts from 0 hr, the 6th row contains the start of first section and so on and so forth.

**Conclusion**

The project gave the insight of the Map/Reduce paradigm. A good knowledge to have for the current trend of problems in the software world.