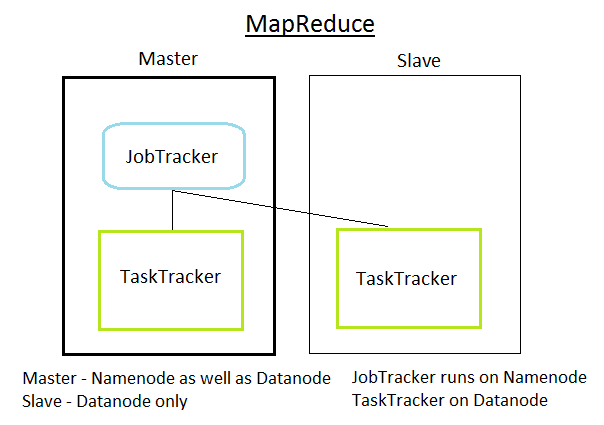
Map Reduce Overview

Let’s have a quick overview about

* Map Reduce is a framework to processes vast amounts of data in parallel on large clusters in reliable and fault tolerant manner.
* It is the computation part of Hadoop architecture and consists of JobTracker and TaskTracker processes.
* There is only one master JobTracker which runs on Namenode and every Datanode will have a slave TaskTracker process. Master is responsible for scheduling the jobs on the slaves, monitoring them and re-executing the failed tasks.
* Map part – Processes the input datasets into multiple chunks. It then sorts the output and then inputs the result to Reduce part.
* Reduce part – It consolidates the output of the Map part.



Map Reduce Lab

In this lab we are going to try the

1. Standard Word count example
2. Modified Word count and search example
3. AirTraffic Data example

In all these examples the generic data flow and steps is as shown in the diagram

# Running Word Count Example

In this example we will run a prebuilt map reduce example from apache hadoop examples library. The word count example will give the count of all the words in the input file.

## Creating input data

* Create a input file

Lets create our own input files. Use standard linux command to create input dir and the input file in input directory

|  |
| --- |
| $ mkdir /home/hadoop/input  $ rm /home/hadoop/input/\*  $ vim /home/hadoop/input/input1.txt |

Add some lines into it

|  |
| --- |
| This is a test input file  Welcome to Big Data Hands on Lab |

Save and close the file

* Copy this input file to hdfs
* Create hdfs input dir

|  |
| --- |
| $ hadoop fs -mkdir /home/hadoop/dfsinput |

For this lab we will use the /home/hadoop/dfsinput as fs input location.

* Copy the input1.txt file to hdfs

|  |
| --- |
| $ hadoop fs -copyFromLocal /home/hadoop/input/input1.txt /home/hadoop/dfsinput |

* Verify the hdfs input file

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/dfsinput/input1.txt  This is a test input file  Welcome to Big Data Hands on Lab |

## Running Word Count Job

* The word count example jar file is

|  |
| --- |
| $ ls $HADOOP\_HOME/hadoop-examples-1.0.3.jar |

* The syntax to execute the job is

|  |
| --- |
| Syntax:  $HADOOP\_HOME/bin/hadoop jar <jar name> <class name> <dfs input dir> <dfs output dir>  Command:  $ hadoop jar $HADOOP\_HOME/hadoop-examples-1.0.3.jar wordcount /home/hadoop/dfsinput /home/hadoop/dfsoutput  Output:  Warning: $HADOOP\_HOME is deprecated.  12/10/21 20:41:16 INFO input.FileInputFormat: Total input paths to process : 1  12/10/21 20:41:16 INFO util.NativeCodeLoader: Loaded the native-hadoop library  12/10/21 20:41:16 WARN snappy.LoadSnappy: Snappy native library not loaded  12/10/21 20:41:16 INFO mapred.JobClient: Running job: job\_201210210947\_0002  12/10/21 20:41:17 INFO mapred.JobClient: map 0% reduce 0%  12/10/21 20:41:34 INFO mapred.JobClient: map 100% reduce 0%  12/10/21 20:41:46 INFO mapred.JobClient: map 100% reduce 100%  12/10/21 20:41:51 INFO mapred.JobClient: Job complete: job\_201210210947\_0002  12/10/21 20:41:51 INFO mapred.JobClient: Counters: 29  12/10/21 20:41:51 INFO mapred.JobClient: Job Counters  12/10/21 20:41:51 INFO mapred.JobClient: Launched reduce tasks=1  12/10/21 20:41:51 INFO mapred.JobClient: SLOTS\_MILLIS\_MAPS=14403  12/10/21 20:41:51 INFO mapred.JobClient: Total time spent by all reduces waiting after reserving slots (ms)=0  12/10/21 20:41:51 INFO mapred.JobClient: Total time spent by all maps waiting after reserving slots (ms)=0  12/10/21 20:41:51 INFO mapred.JobClient: Launched map tasks=1  12/10/21 20:41:51 INFO mapred.JobClient: Data-local map tasks=1  12/10/21 20:41:51 INFO mapred.JobClient: SLOTS\_MILLIS\_REDUCES=10280  12/10/21 20:41:51 INFO mapred.JobClient: File Output Format Counters  12/10/21 20:41:51 INFO mapred.JobClient: Bytes Written=85  12/10/21 20:41:51 INFO mapred.JobClient: FileSystemCounters  12/10/21 20:41:51 INFO mapred.JobClient: FILE\_BYTES\_READ=143  12/10/21 20:41:51 INFO mapred.JobClient: HDFS\_BYTES\_READ=181  12/10/21 20:41:51 INFO mapred.JobClient: FILE\_BYTES\_WRITTEN=43745  12/10/21 20:41:51 INFO mapred.JobClient: HDFS\_BYTES\_WRITTEN=85  12/10/21 20:41:51 INFO mapred.JobClient: File Input Format Counters  12/10/21 20:41:51 INFO mapred.JobClient: Bytes Read=59  12/10/21 20:41:51 INFO mapred.JobClient: Map-Reduce Framework  12/10/21 20:41:51 INFO mapred.JobClient: Map output materialized bytes=143  12/10/21 20:41:51 INFO mapred.JobClient: Map input records=2  12/10/21 20:41:51 INFO mapred.JobClient: Reduce shuffle bytes=143  12/10/21 20:41:51 INFO mapred.JobClient: Spilled Records=26  12/10/21 20:41:51 INFO mapred.JobClient: Map output bytes=111  12/10/21 20:41:51 INFO mapred.JobClient: CPU time spent (ms)=1390  12/10/21 20:41:51 INFO mapred.JobClient: Total committed heap usage (bytes)=171114496  12/10/21 20:41:51 INFO mapred.JobClient: Combine input records=13  12/10/21 20:41:51 INFO mapred.JobClient: SPLIT\_RAW\_BYTES=122  12/10/21 20:41:51 INFO mapred.JobClient: Reduce input records=13  12/10/21 20:41:51 INFO mapred.JobClient: Reduce input groups=13  12/10/21 20:41:51 INFO mapred.JobClient: Combine output records=13  12/10/21 20:41:51 INFO mapred.JobClient: Physical memory (bytes) snapshot=244547584  12/10/21 20:41:51 INFO mapred.JobClient: Reduce output records=13  12/10/21 20:41:51 INFO mapred.JobClient: Virtual memory (bytes) snapshot=1237282816  12/10/21 20:41:51 INFO mapred.JobClient: Map output records=13 |

## Checking the results

* Check the dfs output dir

|  |
| --- |
| $ hadoop fs -ls /home/hadoop/dfsoutput  Found 3 items  -rw-r--r-- 2 hduser supergroup 0 2012-10-21 20:41 /home/hadoop/dfsoutput/\_SUCCESS  drwxr-xr-x - hduser supergroup 0 2012-10-21 20:41 /home/hadoop/dfsoutput/\_logs  -rw-r--r-- 2 hduser supergroup 85 2012-10-21 20:41 /home/hadoop/dfsoutput/part-r-00000 |

The output is written to part-r-00000 file.

* View the content of the output file

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/dfsoutput/part-r-00000  Output:  Big 1  Data 1  Hands 1  Lab 1  This 1  Welcome 1  a 1  file 1  input 1  is 1  on 1  test 1  to 1 |

* Copy the fs output file to local file

|  |
| --- |
| $ hadoop fs -copyToLocal /home/hadoop/dfsoutput/part-r-00000 /home/hadoop/output/wordcount.out  $ vim /home/hadoop/output/wordcount.out |

## Run job again with more inputs

* Add another input file input2.txt in input folder

|  |
| --- |
| $ vim /home/hadoop/input/input2.txt  Input text  This is a another test input file |

* Copy the input2.txt to dfsinput directory

|  |
| --- |
| $ hadoop fs -copyFromLocal /home/hadoop/input/input2.txt /home/hadoop/dfsinput |

* Run the job deleting the dfs output directory

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/dfsoutput  $ hadoop jar $HADOOP\_HOME/hadoop-examples-1.0.3.jar wordcount /home/hadoop/dfsinput /home/hadoop/dfsoutput |

* Check the output

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/dfsoutput/part-r-00000  Output:  Big 1  Data 1  Hands 1  Lab 1  This 2  Welcome 1  a 2  another 1  file 2  input 2  is 2  on 1  test 2  to 1 |

You will notice the count for ‘This’, ‘file’ and other words is now 2.

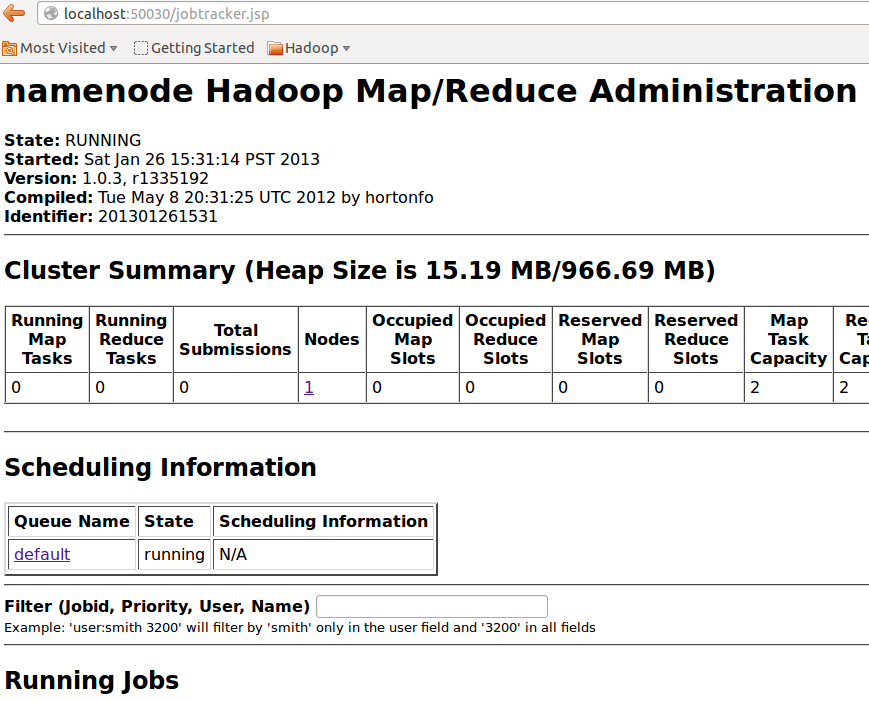
## Various URLs

* Job Tracker

<http://localhost:50030/jobtracker.jsp>

You can view the various jobs that executed, running or retired.

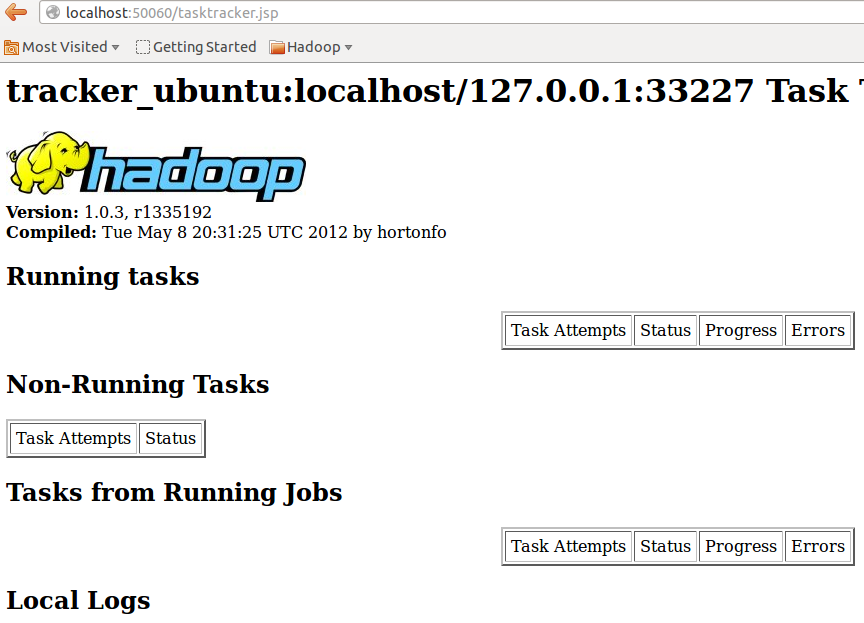
You can also view the logs of the jobs.



* Task Tracker

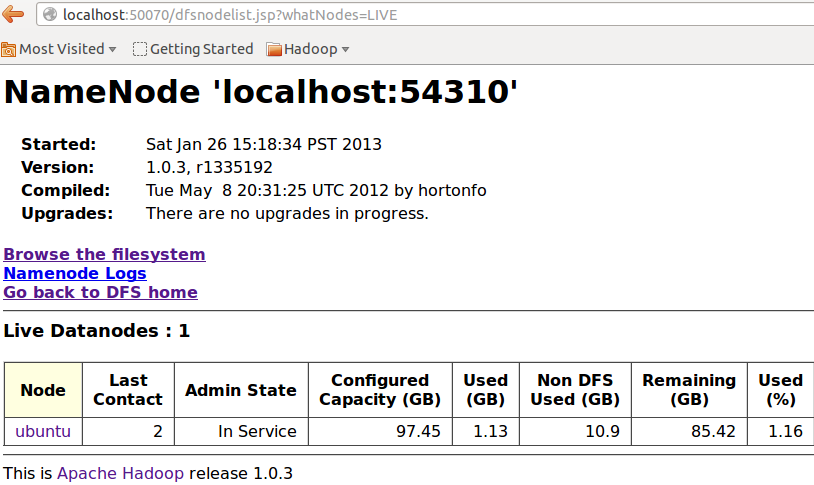
<http://localhost:50060/tasktracker.jsp>

The task tracker URL is relative to each data nodes where the tasktracker is running.



* HDFS Health Check

<http://localhost:50070/dfshealth.jsp>



## Word Count and Search example

WordCountSearch.java

This is slightly modified word count example in which in addition to counting all the words it can search and count the occurrence of a single input word. Let’s try out this example end to end using the java file.

* Create a input folder /home/hadoop/WordCountSearch and add java file WordCountSearch.java with this content.

The java file is already there in the VM at /home/hadoop/mapreduce\_scripts/WordCountSearch/code/WordCountSearch.java

You can just copy it from there.

|  |
| --- |
| import java.io.IOException;  import java.util.\*;    import org.apache.hadoop.fs.Path;  import org.apache.hadoop.conf.\*;  import org.apache.hadoop.io.\*;  import org.apache.hadoop.mapreduce.\*;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;    public class WordCountSearch {    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {  private final static IntWritable one = new IntWritable(1);  private Text word = new Text();    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  Configuration conf = context.getConfiguration();  String inputword = conf.get("searchword");  System.out.println("Input Search Word:"+inputword);  String line = value.toString();  StringTokenizer tokenizer = new StringTokenizer(line);  while (tokenizer.hasMoreTokens()) {  word.set(tokenizer.nextToken());  System.out.println("Word:"+word.toString());  System.out.println("Input Word:"+inputword);  if(!(inputword == null) && !(inputword.equals("ZZZZ"))) {  if(inputword.equals(word.toString())) {  context.write(word, one);  }  } else {  context.write(word, one);  System.out.println("Map element:"+context);  }  }  }  }    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));  }  }    public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  String inputword;  String inputpath;  String outputpath;    System.out.println("Input Arguments");  for(int i=0;i<args.length;i++){  System.out.println("Argument "+i+":"+args[i]);  }  if(args.length == 3){  inputword = args[0];  inputpath = args[1];  outputpath = args[2];  conf.set("searchword",inputword);  System.out.println("Conf get searchword:"+conf.get("searchword"));  } else {  inputword = "No Input Search Word. Counting all words.";  inputpath = args[0];  outputpath = args[1];  conf.set("searchword","ZZZZ");  }  System.out.println("Search word:"+inputword);  System.out.println("Input Path:"+inputpath);  System.out.println("Output Path:"+outputpath);    Job job = new Job(conf, "wordcountsearch");  FileInputFormat.addInputPath(job, new Path(inputpath));  FileOutputFormat.setOutputPath(job, new Path(outputpath));  job.setJarByClass(WordCountSearch.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);    job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);    job.setInputFormatClass(TextInputFormat.class);  job.setOutputFormatClass(TextOutputFormat.class);    job.waitForCompletion(true);  }    } |

## Compile and Run

* Compile the java code

|  |
| --- |
| $ cd /home/hadoop/WordCountSearch  $ javac -classpath $HADOOP\_HOME/hadoop-core-1.0.3.jar:$HADOOP\_HOME/hadoop-client-1.0.3.jar WordCountSearch.java |

* Check the classes

|  |
| --- |
| $ ls /home/hadoop/WordCountSearch  WordCountSearch.class WordCountSearch$Map.class  WordCountSearch$Reduce.class |

* Create jar using the compiled classes

|  |
| --- |
| $ cd /home/hadoop/WordCountSearch  $ jar cvf WordCountSearch.jar WordCountSearch\*.class  added manifest  adding: WordCountSearch.class(in = 2533) (out= 1278)(deflated 49%)  adding: WordCountSearch$Map.class(in = 2700) (out= 1187)(deflated 56%)  adding: WordCountSearch$Reduce.class(in = 1645) (out= 691)(deflated 57%) |

* Verify the jar file

|  |
| --- |
| $ jar tvf /home/hadoop/WordCountSearch/WordCountSearch.jar  0 Sat Oct 27 18:46:40 PDT 2012 META-INF/  68 Sat Oct 27 18:46:40 PDT 2012 META-INF/MANIFEST.MF  2533 Sat Oct 27 18:44:42 PDT 2012 WordCountSearch.class  2700 Sat Oct 27 18:44:42 PDT 2012 WordCountSearch$Map.class  1645 Sat Oct 27 18:44:42 PDT 2012 WordCountSearch$Reduce.class |

* Run the job

We will use the same dfs input directory and input files input1.txt input2.txt

Lets remove the output directory and run the job.

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/dfsoutput  Deleted hdfs://namenode:54310/home/hadoop/dfsoutput  $ hadoop jar /home/hadoop/WordCountSearch/WordCountSearch.jar WordCountSearch /home/hadoop/dfsinput /home/hadoop/dfsoutput  The run log have additional messages printed from custom WordCountSearch.java  Input Arguments  Argument 0:/home/hadoop/dfsinput  Argument 1:/home/hadoop/dfsoutput  Search word:No Input Search Word. Counting all words.  Input Path:/home/hadoop/dfsinput  Output Path:/home/hadoop/dfsoutput |

* Check the output

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/dfsoutput/part-r-00000  Big 1  Data 1  Hands 1  Lab 1  This 2  Welcome 1  a 2  another 1  file 2  input 2  is 2  on 1  test 2  to 1 |

* In this customized Word Count example we can search for a specific word and its count. The program accepts that word as command line argument.

|  |
| --- |
| $ hadoop jar /home/hadoop/WordCountSearch/WordCountSearch.jar WordCountSearch **<input word>** /home/hadoop/dfsinput /home/hadoop/dfsoutput |

* Run the program again to search for input word. First lets delete the dfs output dir

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/dfsoutput  Deleted hdfs://namenode:54310/home/hadoop/dfsoutput  $ hadoop jar /home/hadoop/WordCountSearch/WordCountSearch.jar WordCountSearch **Lab** /home/hadoop/dfsinput /home/hadoop/dfsoutput  Additional output messages  Input Arguments  Argument 0:Lab  Argument 1:/home/hadoop/dfsinput  Argument 2:/home/hadoop/dfsoutput  Conf get searchword:Lab  Search word:Lab  Input Path:/home/hadoop/dfsinput  Output Path:/home/hadoop/dfsoutput |

* Check the output

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/dfsoutput/part-r-00000  Output:  Lab 1 |

## Map Reduce Example using Air Traffic Data

Let’s use some real life data and run the map reduce program to compute valuable information.

The AirTraffic data and the Problem statement is already stated in the ‘1-Initial Setup.doc’

**Data**

The downloaded data file can be found at

|  |
| --- |
| $ /home/hadoop/data/air\_traffic\_jan\_2012.csv |

**Problem**

To compute the average Arrival/Departure delay for various airlines based on certain criteria.

Example: Compute Average Arrival Delay with Origin as New York and Destination Los Angeles for the date 2012-01-01 for all carriers.

**Understanding the data**

Refer to the sample data xls uploaded to the shared google doc folder.

The data has following columns

|  |
| --- |
| Year,Quarter,Month,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Carrier,TailNum,FlightNum,OriginAirportID,OriginAirportSeqID,OriginCityMarketID,Origin,OriginCityName,OriginState,OriginStateFips,OriginStateName,OriginWac,DestAirportID,DestAirportSeqID,DestCityMarketID,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,CRSDepTime,DepTime,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime,ArrDelay,ArrDelayMinutes,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay,FirstDepTime,TotalAddGTime,LongestAddGTime,DivAirportLandings,DivReachedDest,DivActualElapsedTime,DivArrDelay,DivDistance,Div1Airport,Div1AirportID,Div1AirportSeqID,Div1WheelsOn,Div1TotalGTime,Div1LongestGTime,Div1WheelsOff,Div1TailNum,Div2Airport,Div2AirportID,Div2AirportSeqID,Div2WheelsOn,Div2TotalGTime,Div2LongestGTime,Div2WheelsOff,Div2TailNum,Div3Airport,Div3AirportID,Div3AirportSeqID,Div3WheelsOn,Div3TotalGTime,Div3LongestGTime,Div3WheelsOff,Div3TailNum,Div4Airport,Div4AirportID,Div4AirportSeqID,Div4WheelsOn,Div4TotalGTime,Div4LongestGTime,Div4WheelsOff,Div4TailNum,Div5Airport,Div5AirportID,Div5AirportSeqID,Div5WheelsOn,Div5TotalGTime,Div5LongestGTime,Div5WheelsOff,Div5TailNum  Key columns for this exercise are  ArrDelayMinutes  DepDelayMinutes  Origin  Destination  FlightDate  Carrier |

**Map Reduce code**

We have written a Map Reduce program for this lab to solve the above problem.

The java program is AirTraffic.java and can be found at

|  |
| --- |
| /home/hadoop/mapreduce\_scripts/AirTraffic/code/AirTraffic.java |

**Program Input**

Program - AirTraffic.java

To compute the departure delay for American Airline with Origin as Los Angeles and Destination as JFK, New York we need to get this input from the user from command line.

The job takes input in key value pair format. For the example above the input param will be

DepartureDelay:Origin=LAX,Dest=JFK,Carrier=AA

**Run Job**

The program takes max four input key value pair to compute the Arrival/Departure delay.

1. Compute Average Departure Delay for all with Origin as Los Angeles

|  |
| --- |
| DepartureDelay:Origin=LAX |

1. Compute Average Departure Delay for all with Origin as Los Angeles, Destination as New York for all carrier

|  |
| --- |
| DepartureDelay:Origin=LAX,Dest=JFK,Carrier=All |

1. Compute Average Arrival Delay with Origin as New York and Destination Los Angeles for the date 2012-01-01 for all carrier

|  |
| --- |
| ArrivalDelay:Origin=JFK,Dest=LAX,Carrier=All,FlightDate=2012-01-01 |

The steps to run are same as we did for WordCountSearch example

* Java Program AirTraffic.java
* Create an input folder

|  |
| --- |
| $ mkdir /home/hadoop/AirTraffic |

* Copy the AirTraffice.java file to it

|  |
| --- |
| $ cp /home/hadoop/code/AirTraffic.java /home/hadoop/AirTraffic/AirTraffic.java |

* Compile the code

|  |
| --- |
| cd /home/hadoop/AirTraffic  $ javac -classpath $HADOOP\_HOME/hadoop-core-1.0.3.jar:$HADOOP\_HOME/hadoop-client-1.0.3.jar AirTraffic.java |

* Check the classes

|  |
| --- |
| $ ls /home/hadoop/AirTraffic  AirTraffic.class AirTraffic$Map.class AirTraffic$Reduce.class |

* Create jar using the compiled classes

|  |
| --- |
| $ cd /home/hadoop/AirTraffic  $ jar cvf AirTraffic.jar \*.class  added manifest  adding: AirTraffic.class(in = 2533) (out= 1278)(deflated 49%)  adding: AirTraffic$Map.class(in = 2700) (out= 1187)(deflated 56%)  adding: AirTraffic$Reduce.class(in = 1645) (out= 691)(deflated 57%) |

* Verify the jar file

|  |
| --- |
| $ jar tvf /home/hadoop/AirTraffic/AirTraffic.jar  0 Sat Oct 27 18:46:40 PDT 2012 META-INF/  68 Sat Oct 27 18:46:40 PDT 2012 META-INF/MANIFEST.MF  2533 Sat Oct 27 18:44:42 PDT 2012 AirTraffic.class  2700 Sat Oct 27 18:44:42 PDT 2012 AirTraffic$Map.class  1645 Sat Oct 27 18:44:42 PDT 2012 AirTraffic$Reduce.class |

* Job input params

|  |
| --- |
| air-traffic - Input dfs location for the Air Traffic data  dfsoutput – Output dfs location  AirTraffic – Job Name  Command Line parameter for the Map Reduce Job  DepartureDelay:Origin=LAX,Dest=JFK,Carrier=All |

* Run the job
* Input hdfs data

The data prepared in the data section is already present in VM in hdfs format at

|  |
| --- |
| /home/hadoop/dfs-data/air\_traffic |

* Create output hdfs location if not already there

|  |
| --- |
| $ hadoop fs –mkdir /home/hadoop/dfsoutput |

* Run the job

1. DepartureDelay:Origin=JFK,Dest=LAX,Carrier=All,FlightDate=2012-01-01

|  |
| --- |
| $ hadoop jar /home/hadoop/AirTraffic/AirTraffic.jar AirTraffic DepartureDelay:Origin=JFK,Dest=LAX,Carrier=All,FlightDate=2012-01-01  /home/hadoop/dfs-data/air\_traffic /home/hadoop/dfsoutput  Result:  "AA" 16.777779  "B6" 0.0  "DL" 1.6  "UA" 0.0  "VX" 14.0 |

1. ArrivalDelay:Origin=JFK,Dest=LAX,Carrier=All,FlightDate=2012-01-01

|  |
| --- |
| $ hadoop jar /home/hadoop/AirTraffic/AirTraffic.jar AirTraffic ArrivalDelay:Origin=JFK,Dest=LAX,Carrier=All,FlightDate=2012-01-01  /home/hadoop/dfs-data/air\_traffic /home/hadoop/dfsoutput  Result:  "AA" 12.222222  "B6" 1.5  "DL" 3.8  "UA" 0.0  "VX" 4.4 |