**README FILE**

OUTPUT FORMAT: the output files are in the form of (TMEAN\_MIN),(TMEAN\_MAX) instead of the other way around.

Also, the secondary\_sort program takes A FOLDER as input and NOT A FILE. So in the first 3 programs we have two arguments:input and output.

The secondary sort will have FOLDER as input and output directory.

­In order to run the standalone application, open Makefile and change all the variables before “# Pseudo-Cluster Execution” namely hadoop root directory, job.name(specified inside driver class of java file), and jar name specified in pom.xml file where artifactId and groupId form the jar name.

Then open command prompt and install maven if not installed by typing:

sudo apt-get install maven

Then we navigate to the directory where both the java files and the makefile is specified. And then we type:

make alone

Hence the project will compile and give the output.

To run the java files on AWS, open MakeFile and under “# AWS EMR Execution”:

aws.emr.release=emr-5.2.1(version of emr)

aws.region=us-east-1(specify the region under AWS)

aws.bucket.name=mrassignment2(specify the bucket name in S3 directory)

aws.subnet.id=subnet-6356553a

aws.input=s3://mrassignment2/input/1991.csv(specify the input file from S3)

aws.output=s3://mrassignment2/output/output1(specify the output file from S3)

aws.log.dir=s3://mrassignment2/input/log1((specify the log file from S3))

aws.num.nodes=6(specify the number of machines to run the program on)

aws.instance.type=m4.large(specify the type of machine to run it on)

Then we navigate to the directory where both the java files and the makefile is specified. And then we type:

make cloud

and that will create a cluster and give the output in S3. So we navigate to s3 and see the output file after successful termination of the program.

If that does not work,

In order to run the java on AWS, you need to go to Eclipse and make a new maven project. We specify the code in pom.xml given in the snapshot below. Then inside the project make a new package inside src/main/java and then put the java file along with the input inside the package. We also specify log4j.properties file in src/main/resources. The code for that is given below. Then, go to File->Export->Java->JAR file->(Select the project) and then click finish.

Then in AWS we put the JAR file inside our bucket along with the input file in the input directory. Then we create cluster using AWS using the documentation of EMR in HW1 and then we get the output folder in the S3 folder.

To run the standalone program you create a maven project inside Eclipse and inside the pom.xml file, we insert this:

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>2.4.3</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce</artifactId>

<version>2.7.3</version>

<type>pom</type>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-common</artifactId>

<version>2.7.3</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-core</artifactId>

<version>2.7.3</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-jobclient</artifactId>

<version>2.7.3</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>jdk.tools</groupId>

<artifactId>jdk.tools</artifactId>

<version>1.7.0\_05</version>

<scope>system</scope>

<systemPath>${JAVA\_HOME}/lib/tools.jar</systemPath>

</dependency>

</dependencies>

LOG4J.PROPERTIES CODE

log4j.rootLogger=DEBUG, CA

log4j.appender.CA=org.apache.log4j.ConsoleAppender

log4j.appender.CA.layout=org.apache.log4j.PatternLayout

log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

Inside the <project></project> tags.

Then we do Maven clean and build on the project to sort out any dependencies.

We also give arguments to the Program if needed by going to properties->Java Application. In the arguments tab we write the argument that we need to pass it into the function.

Then we run the program.