Module 5: PIG

**Assignment** – Calculate Maximum Temperature

Code:

**package** in.edureka.mapreduce;

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**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** MaxTemp {

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

//Mapper

/\*\*

\* **@method** map

\* <p>This method takes the input as text data type and tokenizes input

\* by taking whitespace as delimiter. The first token goes year and second token is temperature,

\* this is repeated till last token. Now key value pair is made and passed to reducer.

\* **@method\_arguments** key, value, output, reporter

\* **@return** void

\*/

//Defining a local variable k of type Text

Text k= **new** Text();

/\*

\* (non-Javadoc)

\* @see org.apache.hadoop.mapred.Mapper#map(java.lang.Object, java.lang.Object, org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)

\*/

@Override

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

//Converting the record (single line) to String and storing it in a String variable line

String line = value.toString();

//StringTokenizer is breaking the record (line) according to the delimiter whitespace

StringTokenizer tokenizer = **new** StringTokenizer(line," ");

//Iterating through all the tokens and forming the key value pair

**while** (tokenizer.hasMoreTokens()) {

//The first token is going in year variable of type string

String year= tokenizer.nextToken();

k.set(year);

//Takes next token and removes all the whitespaces around it and then stores it in the string variable called temp

String temp= tokenizer.nextToken().trim();

//Converts string temp into integer v

**int** v = Integer.*parseInt*(temp);

//Sending to output collector which inturn passes the same to reducer

context.write(k,**new** IntWritable(v));

}

}

}

//Reducer

/\*\*

\* **@author** sriram!

\* **@interface** Reducer

\* <p>Reduce class is static and extends MapReduceBase and implements Reducer

\* interface having four hadoop generics type Text, IntWritable, Text, IntWritable.

\*/

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

/\*\*

\* **@method** reduce

\* <p>This method takes the input as key and list of values pair from mapper, it does aggregation

\* based on keys and produces the final output.

\* **@method\_arguments** key, values, output, reporter

\* **@return** void

\*/

/\*

\* (non-Javadoc)

\* @see org.apache.hadoop.mapred.Reducer#reduce(java.lang.Object, java.util.Iterator, org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)

\*/

@Override

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

**throws** IOException, InterruptedException {

/\*

\* Iterates through all the values available with a key and if the integer variable temperature

\* is greater than maxtemp, then it becomes the maxtemp

\*/

//Defining a local variable maxtemp of type int

**int** maxtemp=0;

**for**(IntWritable it : values) {

//Defining a local variable temperature of type int which is taking all the temperature

**int** temperature= it.get();

**if**(maxtemp<temperature)

{

maxtemp =temperature;

}

}

//Finally the output is collected as the year and maximum temperature corresponding to that year

context.write(key, **new** IntWritable(maxtemp));

}

}

//Driver

/\*\*

\* **@method** main

\* <p>This method is used for setting all the configuration properties.

\* It acts as a driver for map reduce code.

\* **@return** void

\* **@method\_arguments** args

\* **@throws** Exception

\*/

**public** **static** **void** main(String[] args) **throws** Exception {

//reads the default configuration of cluster from the configuration xml files

Configuration conf = **new** Configuration();

//Initializing the job with the default configuration of the cluster

Job job = **new** ~~Job~~(conf, "MaxTemp");

//Assigning the driver class name

job.setJarByClass(MaxTemp.**class**);

//Defining the mapper class name

job.setMapperClass(Map.**class**);

//Defining the reducer class name

job.setReducerClass(Reduce.**class**);

//Defining the output key class for the final output i.e. from reducer

job.setOutputKeyClass(Text.**class**);

//Defining the output value class for the final output i.e. from reducer

job.setOutputValueClass(IntWritable.**class**);

//Defining input Format class which is responsible to parse the dataset into a key value pair

job.setInputFormatClass(TextInputFormat.**class**);

//Defining output Format class which is responsible to parse the final key-value output from MR framework to a text file into the hard disk

job.setOutputFormatClass(TextOutputFormat.**class**);

//setting the second argument as a path in a path variable

Path outputPath = **new** Path(args[1]);

//Configuring the input/output path from the filesystem into the job

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

//deleting the output path automatically from hdfs so that we don't have delete it explicitly

outputPath.getFileSystem(conf).~~delete~~(outputPath);

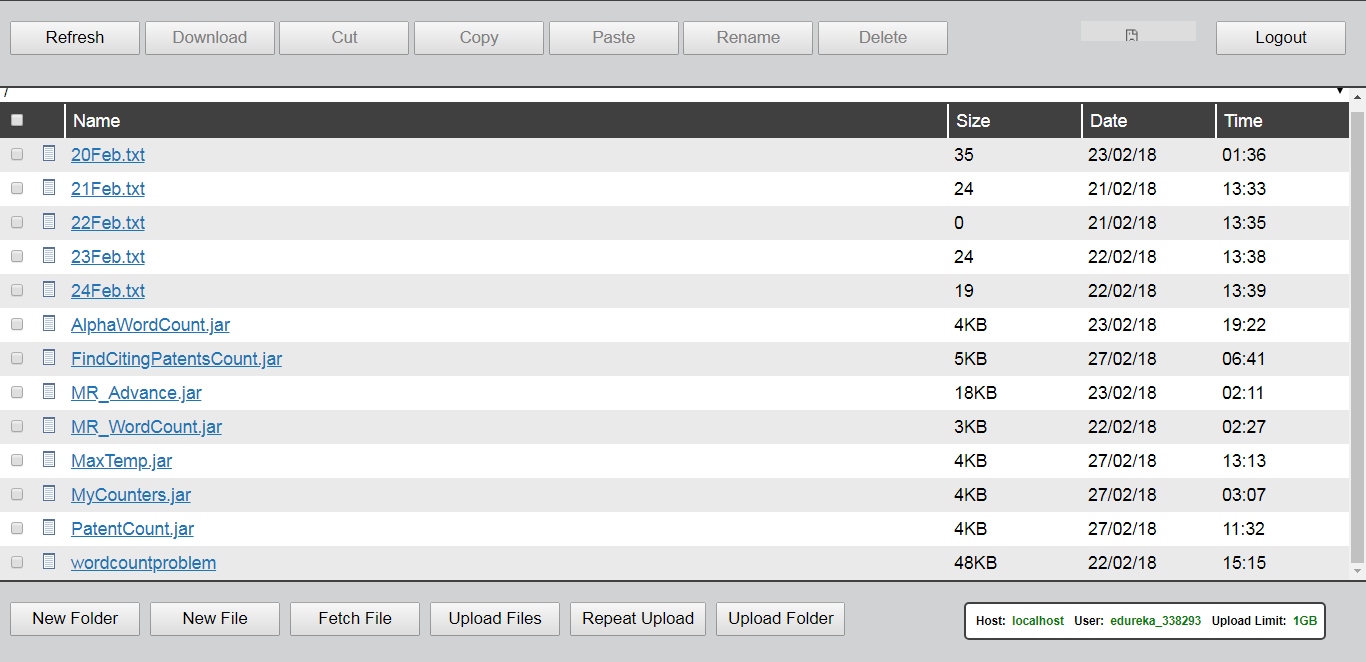
//exiting the job only if the flag value becomes false

System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

}

}

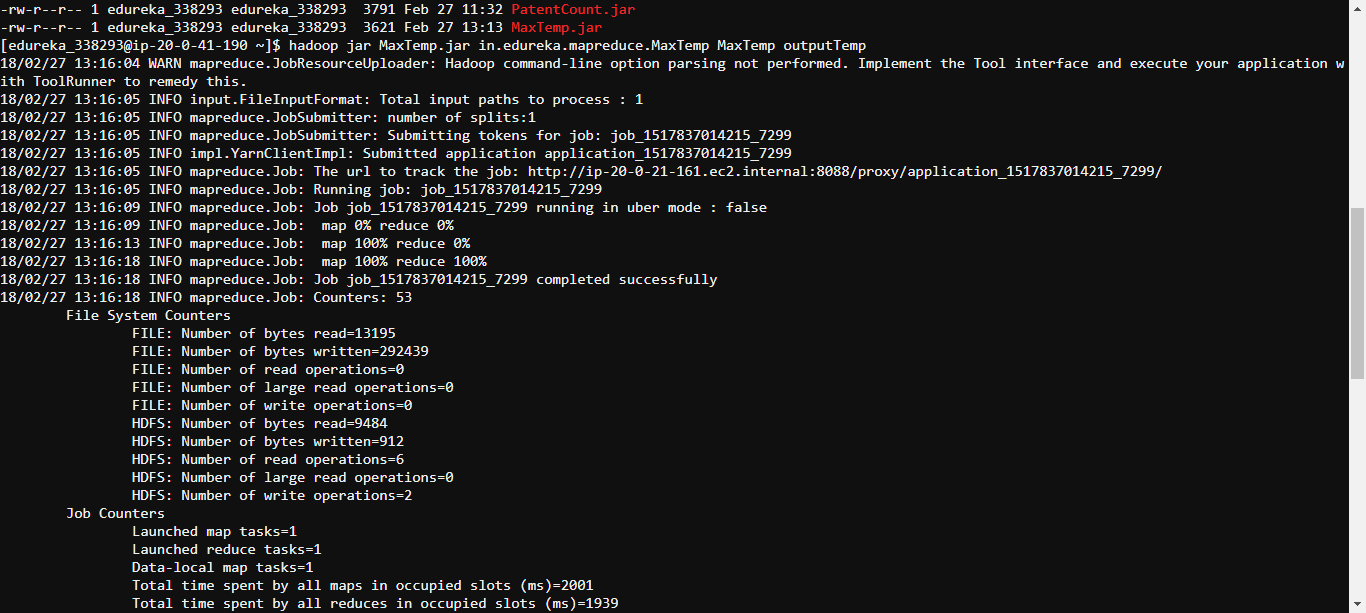
FTP:



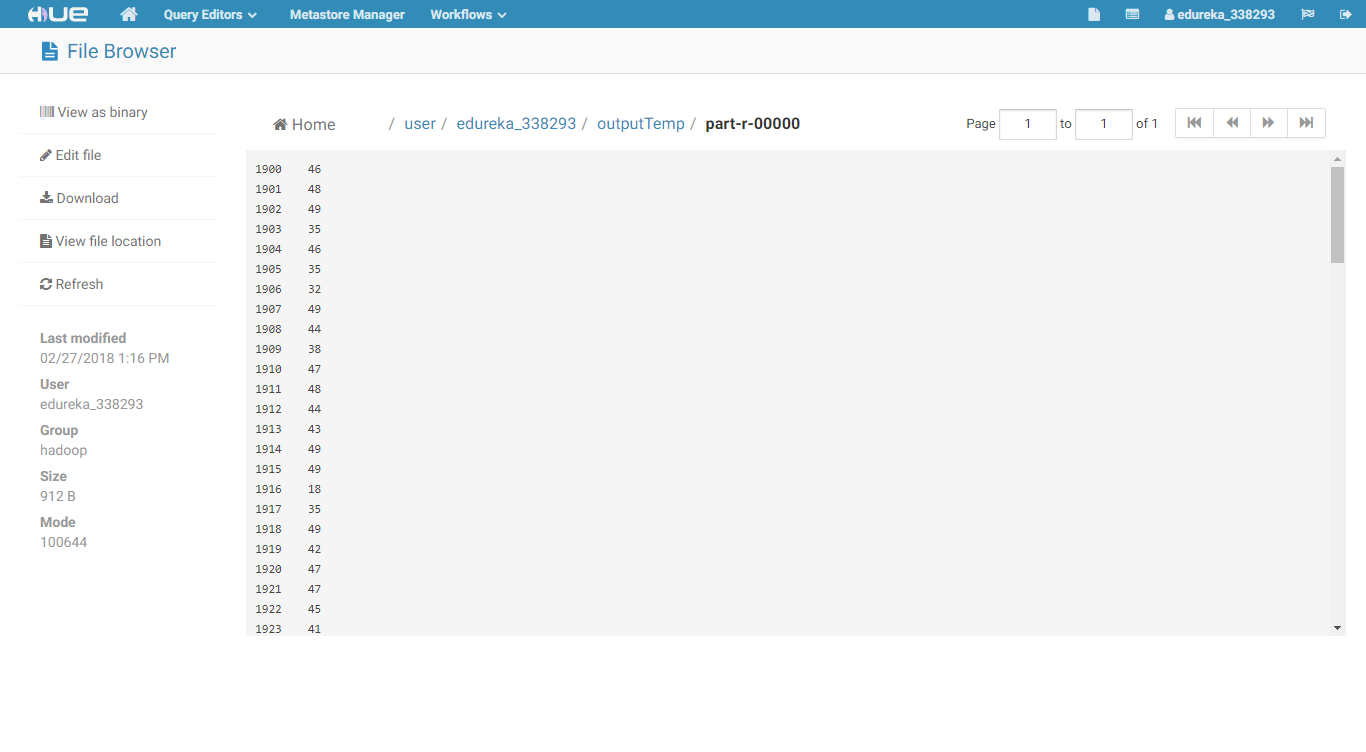
Command:

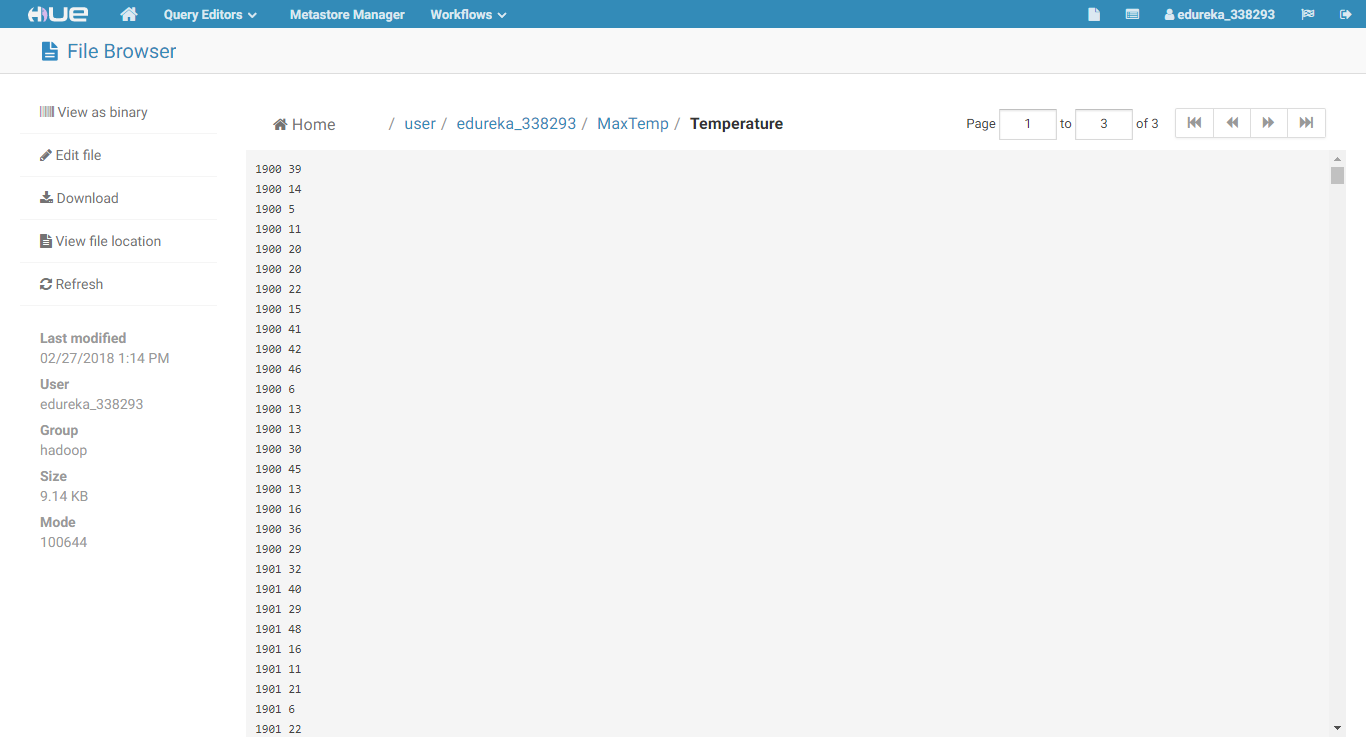
hadoop jar MaxTemp.jar in.edureka.mapreduce.MaxTemp MaxTemp outputTemp

Webconsole:



HUE:





Output:

1900 46

1901 48

1902 49

1903 35

1904 46

1905 35

1906 32

1907 49

1908 44

1909 38

1910 47

1911 48

1912 44

1913 43

1914 49

1915 49

1916 18

1917 35

1918 49

1919 42

1920 47

1921 47

1922 45

1923 41

1924 49

1925 48

1926 49

1927 47

1928 48

1929 35

1930 48

1931 37

1932 33

1933 43

1934 47

1935 40

1936 48

1937 44

1938 43

1939 48

1940 49

1941 49

1942 24

1943 45

1944 39

1945 47

1946 48

1947 41

1948 42

1949 48

1950 41

1951 48

1952 49

1953 43

1954 46

1955 47

1956 48

1957 49

1958 34

1959 45

1960 44

1961 35

1962 46

1963 27

1964 42

1965 35

1966 48

1967 41

1968 47

1969 47

1970 49

1971 42

1972 49

1973 49

1974 49

1975 46

1976 40

1977 49

1978 48

1979 31

1980 48

1981 49

1982 45

1983 38

1984 44

1985 48

1986 48

1987 43

1988 49

1989 47

1990 44

1991 49

1992 48

1993 45

1994 38

1995 48

1996 47

1997 49

1998 48

1999 46

2000 49

2001 45

2002 45

2003 30

2004 48

2005 48

2006 47

2007 47

2008 46

2009 43

2010 47

2011 42

2012 45

2013 49