|  |
| --- |
| 08/07/2018 | By: Ajit K Prasad |



|  |
| --- |
| Big Data Engineering with Hadoop & Spark |
| Case Study I – Movie Data Analysis |

Case Study I – Movie Data Analysis

This assignment is aimed at consolidating the concepts that was learnt during the MapReduce & Apache Pig.

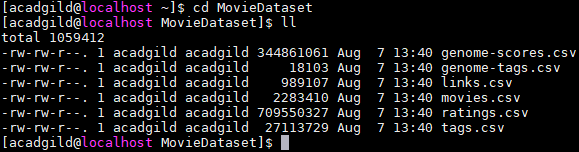
# Problem Statement:

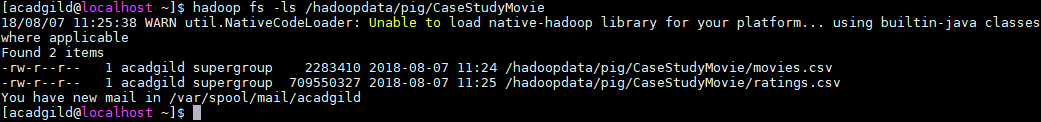
* [Movie datasets](#_Datasets_on_local) were provided for various task to be performed using Pig Latin script both in local and MapReduce mode.
* In [Task 1](#_Task_1:) of this case study, movies that were never rated was also found out.
* Codes were tested on local as well as on HDFS (MR mode). MapReduce Output follows local output.
* The output count has been kept to 10 on purpose, to demonstrate that the logic is correct.
* [Java MapReduce program](#_Java_MapReduce_program) with multiple Mapper was used to process the data as well.

**Note:** Program files are properly documented for a detailed description of each instruction used within the program along with sample inputs.

# Datasets on Local & HDFS:

* Files used for analysis:
  + movies.csv
  + ratings.csv





# Task 1:

* What are the movie titles that the user has rated & not rated?

**Solution: (Local mode)**

* Execute Pig Latin script on ***local*** mode

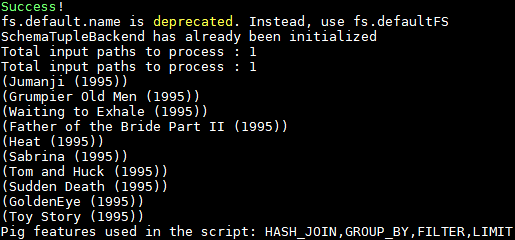
*$* ***pig -brief -x local*** *Query1.pig*

**Command Explanation:**

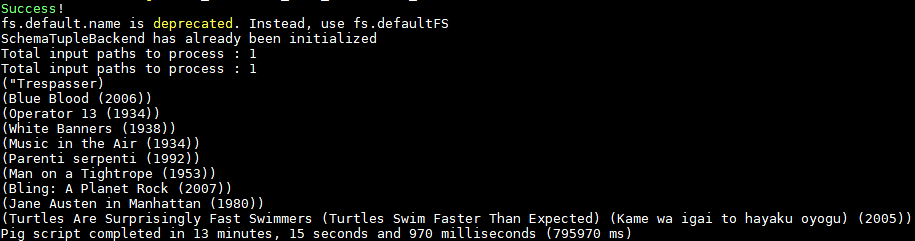
* **pig -x local**: runs pig command in local mode (since it is very large file, running locally)
* **-brief**: ignores unwanted info messages dump over screen

**Output: (Local mode)**

* Rated



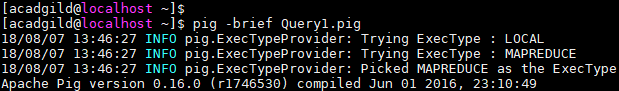
* Not rated



**Solution: (MapReduce mode)**

* Execute Pig Latin script on ***MapReduce*** mode

*$* ***pig -brief*** *Query1.pig*

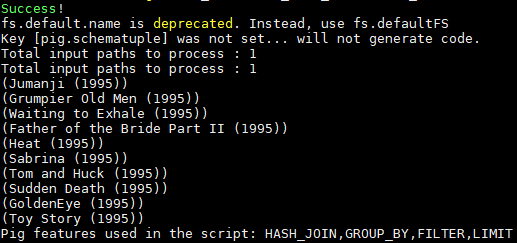


**Command Explanation:**

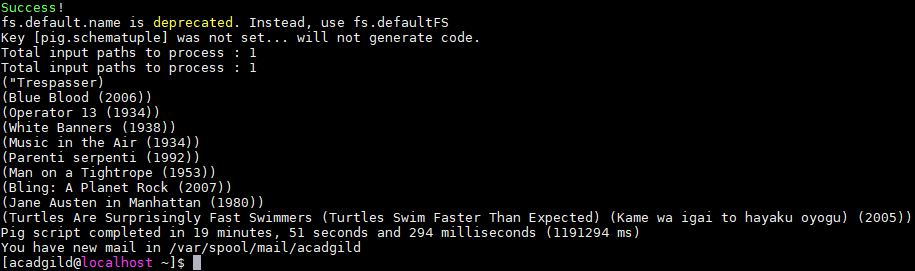
* **pig -x local**: runs pig script in MR mode (this is the default mode)
* **-brief**: ignores unwanted info messages dump over screen

**Output: (MapReduce mode)**

* Rated



* Not rated



# Task 2:

* How many times a movie has been rated by the user?

**Solution: (Local mode)**

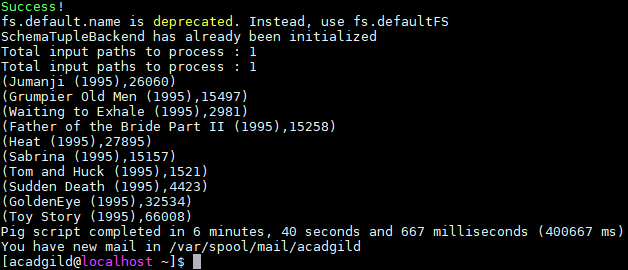
* Execute Pig Latin script on ***local*** mode

*$* ***pig -brief -x local*** *Query2.pig*

**Command Explanation:**

* **pig -x local**: runs pig command in local mode (since it is very large file, running locally)
* **-brief**: ignores unwanted info messages dump over screen

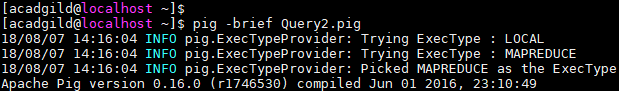
**Output: (Local mode)**



**Solution: (MapReduce mode)**

* Execute Pig Latin script on ***MapReduce*** mode

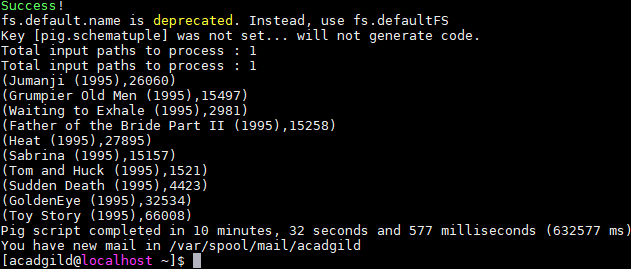
*$* ***pig -brief*** *Query2.pig*



**Command Explanation:**

* **pig -x local**: runs pig script in MR mode (this is the default mode)
* **-brief**: ignores unwanted info messages dump over screen

**Output: (MapReduce mode)**



# Task 3:

* What is the average rating given for a movie?

**Solution: (Local mode)**

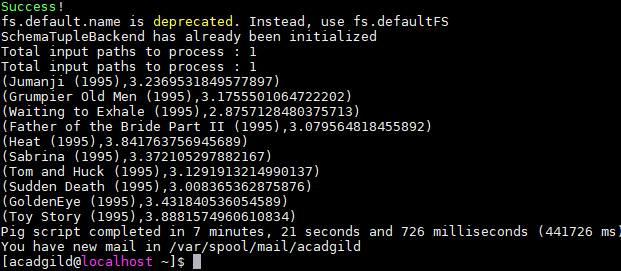
* Execute Pig Latin script on ***local*** mode

*$* ***pig -brief -x local*** *Query3.pig*

**Command Explanation:**

* **pig -x local**: runs pig command in local mode (since it is very large file, running locally)
* **-brief**: ignores unwanted info messages dump over screen

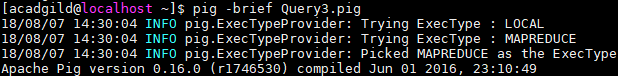
**Output: (Local mode)**



**Solution: (MapReduce mode)**

* Execute Pig Latin script on ***MapReduce*** mode

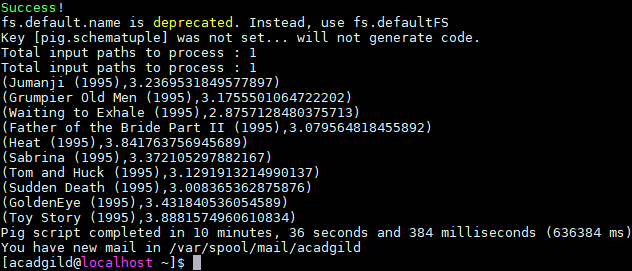
*$* ***pig -brief*** *Query3.pig*



**Command Explanation:**

* **pig -x local**: runs pig script in MR mode (this is the default mode)
* **-brief**: ignores unwanted info messages dump over screen

**Output: (MapReduce mode)**



# Java MapReduce program

* What are the movie titles that the user has rated?
* How many times a movie has been rated by the user?
* In question 2 above, what is the average rating given for a movie?

**RATING MAPPER**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class CaseStudyIUseCasesRatingsMapper extends

Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

try {

if (key.get() == 0 && value.toString().contains("userId")){

return;

} else {

String record = value.toString();

String[] parts = record.split(",");

context.write(new Text(parts[1]), new Text("ratings\t" +

parts[2]));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Explanation:**

This code is to map the rating:

* Here we are checking the input received from input and files and bifurcating them accordingly
* Input values are LongWritable and text formats while outputs are in Text formats
* We are taking only UserID & rating from this file
* We are checking if key and values are null, then return. If not split the inputs by “,” and parts[1] in the parts array is UserID and parts[2] is movierating
* This UserID i.e. Key and rating i.e. Value is sent as output to the reducer from this mapper

**MOVIE MAPPER**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class CaseStudyIUseCasesMoviesMapper extends

Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

try {

if (key.get() == 0 && value.toString().contains("movieId")){

return;

} else {

String record = value.toString();

String[] parts = record.split(",");

context.write(new Text(parts[0]), new Text("movies\t" + parts[1]));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Explanation:**

This code is to map the rating:

* Here we are checking the input received from input and files and bifurcating them accordingly
* Input values are LongWritable and text formats while outputs are in Text formats
* We are taking only movieID & moviename from this file
* We are checking if key and values are null, then return. If not split the inputs by “,” and parts[0] in the parts array is movieID and parts[1] is moviename
* This movieID i.e. Key and moviename i.e. Value is sent as output to the reducer from this mapper

**REDUCER**

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class CaseStudyIUseCasesReducer extends

Reducer<Text, Text, Text, Text> {

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

throws IOException, InterruptedException {

String titles = "";

double total = 0.0;

int count = 0;

System.out.println("Text Key =>"+key.toString());

for (Text t : values) {

String parts[] = t.toString().split(",");

System.out.println("Text values =>"+t.toString());

if (parts[0].equals("ratings")) {

count++;

String rating = parts[1].trim();

System.out.println("Rating is =>"+rating);

total += Double.parseDouble(rating);

} else if (parts[0].equals("movies")) {

titles = parts[1];

} }

double average = total / count;

String str = String.format("%d\t%f", count, average);

context.write(new Text(titles), new Text(str));

}

}

**Explanation:**

* Here outputs of two mappers are inputs to this reducer
* Both input and outputs are Text format
* Now we check all the inputs and bifurcate them accordingly.
* UserID and MovieID are the keys, we split the input by “,” and check if the part is “rating” or not
  + If the part is rating then we print the rating and calculate the total
  + If the part is not rating then it must moviename, then we pring the moviename and save it in the variable “title”
* We calculate the average of the rating for a particular movie title
* We print the number of times the movie was rating by the user and the average rating

**DRIVER**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CaseStudyIUseCasesDriver {

@SuppressWarnings("deprecation")

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

if (args.length != 3) {

System.err.println("Usage: CaseStudyIUseCase2Driver <input path1> <input path2> <output path>");

System.exit(-1);

}

//Job Related Configurations

Configuration conf = new Configuration();

Job job = new Job(conf, "CaseStudyIUseCase2Driver");

job.setJarByClass(CaseStudyIUseCasesDriver.class);

//job.setNumReduceTasks(0);

//Since there are multiple input, there is a slightly different way of specifying input path,

input format and mapper

MultipleInputs.addInputPath(job, new Path(args[0]),TextInputFormat.class,

CaseStudyIUseCasesMoviesMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]),TextInputFormat.class,

CaseStudyIUseCasesRatingsMapper.class);

//Set the reducer

job.setReducerClass(CaseStudyIUseCasesReducer.class);

//set the out path

Path outputPath = new Path(args[2]);

FileOutputFormat.setOutputPath(job, outputPath);

outputPath.getFileSystem(conf).delete(outputPath, true);

//set up the output key and value classes

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

//execute the job

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

}

}

**Explanation:**

* Here there are 2 input paths and 1 output path, thereby, we check if all the 3 parameters are entered by the user, if not an error is given saying user has to enter 3 parameters and exits
* Job configuration instance is created and driverclass is set jar by class
* Multiple input path are defined under args[0] and args[1], as we have two csv files. So, each csv file is given in two different paths
* Output path is defined and also output key and value class

**Command:**

*$ hadoop jar CaseStudyI.jar /hadoopdata/pig/CaseStudyMovie/movies.csv /hadoopdata/pig/CaseStudyMovie/ratings.csv /hadoopdata/pig/CaseStudyMovie/MROutput*

**Output Screens:**

