UECS3223 Cloud Computing

Jan 2017

Group Project

**Group members**

|  |  |  |
| --- | --- | --- |
| Name | Student ID | Trimester and year of study |
| 1. Chow Ee Ling | 1403712 | Y2T3 |
| 2. Choy Yi Zhen | 1404206 | Y2T3 |
| 3. Lim Ru Sin | 1403537 | Y2T3 |
| 4. Rufina Huong Yi Jing | 1501607 | Y2T3 |

**Marks breakdown**

|  |  |  |
| --- | --- | --- |
| **Parts** | **Marks allocated** | **Comments** |
| a) Proper instructions on how to set up the workflow for the data analysis and / or ETL operations so that they run properly | / 20 |  |
| b) Implementation of MapReduce / Spark applications or Hive / Spark SQL queries that perform AT LEAST 3 types of different analysis on the selected dataset. | / 30 |  |
| c) Complexity of analysis performed; as given by the analysis algorithm implemented in MapReduce / Spark or the queries executed in Hive / Spark SQL | / 20 |  |
| d) Proper explanation of the selected analysis being performed, along with comments in source code to explain how the analysis is achieved. | / 30 |  |
| **Total** | / 100 |  |

1.0 Setup

1.1 Spark

- download data as csv file from this link:

Students' Academic Performance Dataset- <https://www.kaggle.com/aljarah/xAPI-Edu-Data>

- put education.csv file in local directory ‘/home/cloudera’

- open shell terminal in cloudera, put data file from local directory into HDFS directory

hdfs dfs -put education.csv /user/cloudera/

- start spark shell

spark-shell

- read data file from HDFS directory

val datafile = sc.textFile("education.csv")

- remove header from datafile

val rmheader = datafile.mapPartitionsWithIndex {(idx, iter) => if (idx == 0) iter.drop(1) else iter}

- map data into array

val data = rmheader.map(line=>line.split(","))

- view data

data.collect

Output:

Array(Array(M, KW, KuwaIT, lowerlevel, G-04, A, IT, F, Father, 15, 16, 2, 20, Yes, Good, Under-7, M), Array(M, KW, KuwaIT, lowerlevel, G-04, A, IT, F, Father, 20, 20, 3, 25, Yes, Good, Under-7, M), Array(M, KW, KuwaIT, lowerlevel, G-04, A, IT, F, Father, 10, 7, 0, 30, No, Bad, Above-...

1.2 Hive

- Open another terminal, start hive shell

Hive

- Create a database called education, type at the hive shell:

CREATE DATABASE education;

- To show the database is created, type:

SHOW DATABASES;

Output:

default

education

- An existing database must be selected as the current working database before any table related operations are performed, type:

USE education;

- Determining the current database in use, type:

SELECT current\_database();

Output:

education

- Before we view the contents of the table to ensure the contents of the file was correctly uploaded, will set a Hive variable to allow display of column names by typing:

SET hive.cli.print.header = TRUE;

2.0 Analysis

2.1 Analysis 1: Find the subject which has the highest amount of students

There are total 12 topic in the table, this analysis is to find topic that has the highest amount of students. In Spark, we count the number of row for each Topic because one row indicate one student. The result is saved in a text file to be used in Hive. After that, in Hive, we select the row with maximum student number and display the Topic.

2.1.1 Spark

2.1.1.1 Map data into RDD

val topicRDD = data.map(x => (x(6),1)).reduceByKey(\_+\_)

- map() return the topic column with an integer 1 for each topic.

- reduceByKey() group data by topic name and the integer 1 is added up for each topic.

topicRDD.collect

Output:

Array[(String, Int)] = Array((Arabic,59), (Science,51), (Math,21), (French,65), (Spanish,25), (English,45), (Chemistry,24), (History,19), (Geology,24), (IT,95), (Biology,30), (Quran,22))

2.1.1.2 Save RDD into text file in HDFS

topicRDD.map(x => x.\_1 + "," + x.\_2).saveAsTextFile("user/cloudera/topic\_number")

- map() remove the parentheses of the output before saving into text file in HDFS direcrtory.

2.1.2 Hive

2.1.2.1 Create table

Create a table with columns based on the primitive types:

CREATE TABLE topic (topic varchar(255), totalStudents smallint)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Next, load this ‘topic\_number’ file into the table directory that was previously created in Spark by typing at the Hive shell:

LOAD DATA INPATH 'user/cloudera/topic\_number/part-00000' INTO TABLE topic;

LOAD DATA INPATH 'user/cloudera/topic\_number/part-00001' INTO TABLE topic;

And then retrieve all the rows in the table with:

SELECT \* FROM topic;

Output:

topic.topic topic.totalstudents

Arabic 59

Science 51

Math 21

French 65

Spanish 25

English 45

Chemistry 24

History 19

Geology 24

IT 95

Biology 30

Quran 22

2.1.2.2 Create View ‘totalStudents’

Create a view called ‘totalStudents’ that retrieve the highest amount of students from the ‘topic’ table:

CREATE VIEW totalStudents AS

SELECT MAX(totalstudents) AS totalStudents

FROM topic;

And then retrieve all the result in the table with:

SELECT \* FROM totalStudents;

Output:

totalstudents.totalstudents

95

2.1.2.3 Display Analysis Output

In order to obtain the subject which has the highest amount of students, type the following command:

SELECT topic AS subject

FROM topic AS t

INNER JOIN totalStudents AS ts

ON t.totalstudents = ts.totalstudents;

Output:

subject

IT

From the data retrieved, IT is the subject which has the highest amount of students.

2.2 Analysis 2: Find gender that score best in each topic

This analysis is to determine which gender(male or female) get a better grade in each subject.

Dataset given gradeID is a string, therefore, we convert each gradeID into relevant integer. The smaller the grade integer, the better the result (e.g G-01 = 1 = A+). After that, we select column topic, gradeID and gender to save as text file for analysis in Hive later. In Hive, we create two table and join them together so that we have integer as our ‘grade’ instead of string. Another two view ‘mingrade’ and ‘gradeAvg’ is created and joined to display the gender that have smaller average grade for each topic. Topics that both gender(‘M’ & ‘F’) are displayed indicates that both gender have same performance in that topic.

2.2.1 Spark

2.2.1.1 Convert gradeID into integer

val gradeidRDD = data.map(x => x(4)).distinct()

- map() return all the gradeID

- distinct() filter out repetition gradeID

def classifyGrade(grade: String) : (String, Int) = {

grade match {

case "G-01" => (grade, 1)

case "G-02" => (grade, 2)

case "G-03" => (grade, 3)

case "G-04" => (grade, 4)

case "G-05" => (grade, 5)

case "G-06" => (grade, 6)

case "G-07" => (grade, 7)

case "G-08" => (grade, 8)

case "G-09" => (grade, 9)

case "G-10" => (grade, 10)

case "G-11" => (grade, 11)

case "G-12" => (grade, 12)

}

}

val gradeInt = gradeidRDD.map(classifyGrade)

- classifyGrade() return gradeID with its relative integer, the result is stored in gradeInt

gradeInt.collect

Output:

res2: Array[(String, Int)] = Array((G-08,8), (G-06,6), (G-11,11), (G-02,2), (G-04,4), (G-09,9), (G-07,7), (G-05,5), (G-10,10), (G-12,12))

2.2.1.2 Store gradeInt into text file in HDFS

gradeInt.map(x => x.\_1 + "," + x.\_2).saveAsTextFile("user/cloudera/grade\_int")

2.2.1.3 Map data into RDD

val topicGradeGender = data.map(x => (x(6), x(4), x(0)))

- map() return the columns topic, gradeID and gender

topicGradeGender.collect

Output:

Array[(String, String, String)] = Array((IT,G-04,M), (IT,G-04,M), (IT,G-04,M), (IT,G-04,M), (IT,G-04,M), (IT,G-04,F), (Math,G-07,M), (Math,G-07,M), (Math,G-07,F), (IT,G-07,F), (Math,G-07,M..

2.2.1.3 Save RDD into text file

topicGradeGender.map(x => x.\_1 + "," + x.\_2 + "," + x.\_3).saveAsTextFile("user/cloudera/topic\_grade\_gender")

- map() is to remove the parentheses of the output before saving into text file in HDFS direcrtory.

2.2.2 Hive

2.2.2.1 Create table

Create tables with columns based on the primitive types:

CREATE TABLE topic\_grade\_gender(topic varchar(255), grade varchar(255), gender char(1))

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

CREATE TABLE grades(grade varchar(255), intgrade smallint)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Next, load ‘topic\_grade\_gender’ and “grade\_int” file into the table directory that was previously created in Spark by typing at the Hive shell:

LOAD DATA INPATH 'user/cloudera/topic\_grade\_gender/part-00000' INTO TABLE topic\_grade\_gender;

LOAD DATA INPATH 'user/cloudera/topic\_grade\_gender/part-00001' INTO TABLE topic\_grade\_gender;

LOAD DATA INPATH 'user/cloudera/grade\_int/part-00000' INTO TABLE grades;

LOAD DATA INPATH 'user/cloudera/grade\_int/part-00001' INTO TABLE grades;

And then retrieve all the rows in the table with:

SELECT \* FROM grades;

Output:

grades.grade grades.intgrade

G-08 8

G-06 6

G-11 11

G-02 2

G-04 4

G-09 9

G-07 7

G-05 5

G-10 10

G-12 12

SELECT \* FROM topic\_grade\_gender;

Output:

topic\_grade\_gender.topic topic\_grade\_gender.grade topic\_grade\_gender.gender

IT G-04 M

IT G-04 M

IT G-04 M

IT G-04 M

IT G-04 M

IT G-04 F

Math G-07 M

Math G-07 M

Math G-07 F

IT G-07 F

Math G-07 M

Math G-07 M

IT G-04 M

Math G-08 M

Math G-08 F

...

2.2.2.2 Create View ‘gradeAvg’

Create a view called ‘gradeAvg’ that retrieve the subject, average of grade and gender from the ‘topic\_grade\_gender’ and ‘grades’ table:

CREATE VIEW gradeAvg AS

SELECT tgg.topic AS subject, avg(g.intgrade) AS avggrade, tgg.gender AS gender

FROM topic\_grade\_gender AS tgg

INNER JOIN grades AS g

ON g.grade=tgg.grade

GROUP BY tgg.topic, tgg.gender;

And then retrieve all the rows in the table with:

SELECT \* FROM gradeAvg;

Output:

subject avggrade gender

Arabic 5.0 F

Arabic 4.837209302325581 M

Biology 7.0 F

Biology 7.0 M

Chemistry 8.0 F

Chemistry 8.0 M

English 8.210526315789474 F

English 6.615384615384615 M

French 2.3 F

French 2.0 M

Geology 8.0 F

Geology 8.0 M

History 6.0 F

History 6.909090909090909 M

IT 4.15625 F

IT 4.238095238095238 M

Math 7.0 F

Math 7.25 M

Quran 7.777777777777778 F

Quran 7.615384615384615 M

Science 5.3478260869565215 F

Science 6.035714285714286 M

Spanish 8.0 F

Spanish 8.125 M

2.2.2.3 Create View ‘mingrade’

Create a view called ‘mingrade’ that retrieve the minimum grade from the ‘gradeAvg’ view:

CREATE VIEW mingrade AS

SELECT subject, MIN(avggrade) AS minavg FROM gradeAvg

GROUP BY subject;

And then retrieve all the rows in the table with:

SELECT \* FROM mingrade;

Output:

subject minavg

Arabic 4.837209302325581

Biology 7.0

Chemistry 8.0

English 6.615384615384615

French 2.0

Geology 8.0

History 6.0

IT 4.15625

Math 7.0

Quran 7.615384615384615

Science 5.3478260869565215

Spanish 8.0

2.2.2.3 Display Analysis Output

Type the following command:

SELECT mg.subject AS subject, ga.gender AS gender FROM gradeavg AS ga

INNER JOIN mingrade AS mg

ON mg.subject = ga.subject AND avggrade = minavg;

Output:

subject gender

Arabic M

Biology F

Biology M

Chemistry F

Chemistry M

English M

French M

Geology F

Geology M

History F

IT F

Math F

Quran M

Science F

Spanish F

Gender that score best in each topic is retrieved from the table. If the subject display both male female, it indicates the male and female score the same average mark in that subject.

2.3 Analysis 3: For students that absent more than 7 days, find number of student that score each grade for Math

We select column gradeID, topic, studentAbsenceDays and filter it to keep only record with topic "Math" and studentAbsenceDays "Above-7". Again, we count the number of row for each gradeID because one row indicate one student.

2.3.1 Spark

2.3.1.1 Map data into RDD

val mapSection = data.map(word=>(word(4),word(6),word(15)))

- map() return gradeID, topic, studentAbsenceDays

val filterData = mapSection.filter(x=>x.\_3.contains("Above-7") && x.\_2.contains("Math"))

- filter() return students that absent more than 7 days and taking Math topic

val filterColumn = filterData.map(x => (x.\_1))

- map() return gradeID of filterData

val countRDD = filterColumn.map(line=>(line,1))

- map() add an integer 1 for every gradeID

val keepCount = countRDD.reduceByKey((x,y) => x + y)

- reduceByKey() group data by gradeID and the integer 1 is added up for each gradeID.

2.3.1.2 Display Analysis Data

println("The gradeID vs number of students is : \n")

keepCount.foreach(println)

Output :

scala> println("The gradeID vs number of students is : \n")

The gradeID vs number of students is :

scala> keepCount.foreach(println)

(G-08,3)

(G-06,1)

(G-11,1)

(G-04,1)

(G-07,4)

2.4 Analysis 4: Find the average time of visits to a course content for each student in the first semester

We select column semester and visitedResources. If the column semester contains “F”, the sum of initial value and visitedResources will be returned else only the initial value will be returned. Then, all partitions are combined with the initial value being used in the operation. Applying simple addition operation to them gives the final result.

2.4.1 Spark

2.4.1.1 Map data into RDD

val mapData = data.map(line=>(line(7),line(10)))

-map() returns semester, visitedResources

val total = mapData.aggregate(0)((x,y)=>if(y.\_1.contains("F")) x + y.\_2.toInt else x, (a, b) => a + b)

- aggregate() accepts three parameters:

1)(0)- similar to the value in fold(), this is an initialization value used for the initial function application on each partition

2) ((x,y)=>if(y.\_1.contains("F")) x + y.\_2.toInt else x - function that checks which column contains “F” and if true it will return the sum of initial value and visitedResources

3) y.\_2.toInt else x, (a, b) => a + b) - function that combine all partitions by applying simple addition to them to produce the final result

val count = mapData.filter(x => x.\_1.contains("F")).count

- filter() returns all students of the first semester and count the total students

2.4.1.2 Display Analysis Data

println("The average times of visits to a course content is : " + total / count.toDouble)

Output :

total: Int = 12052

count: Long = 245

The average times of visits to a course content is : 49.19183673469388

2.5 Analysis 5: Find the educational stage that students score the best grade

In this analysis, we are going to find each educational stage students that score the best grade. We selected column stageID and gradeID from the data set to save as text file for analysis in Hive later. In Hive, we created a new table called stages\_grade to store the data that selected from Spark. One view is created to find the highest grade(int) from each education stage. After that, the view is joined to the ‘grades’ table to display each education and the stage level(String).

2.5.1 Spark

2.5.1.1 Map data into RDD

val stages = data.map(x=>(x(3),x(4)))

- map.() return stageID and gradeID

stages.collect

Output:

res28: Array[(String, String)] = Array((lowerlevel,G-04), (lowerlevel,G-04), (lowerlevel,G-04), (lowerlevel,G-04), (lowerlevel,G-04), (lowerlevel,G-04), (MiddleSchool,G-07), (MiddleSchool,G-07), (MiddleSchool,G-07), (MiddleSchool,G-07), (MiddleSchool,G-07), (MiddleSchool,G-07...

stages.map(x => x.\_1 + "," + x.\_2).saveAsTextFile("user/cloudera/stages\_grade")

2.5.2 Hive

2.5.2.1 Create table

Create a table with columns based on the primitive types:

CREATE TABLE stages\_grade(stage varchar(255), grade varchar(255))

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Next, load this ‘stages\_grade’ file into the table directory that was previously created in Spark by typing at the Hive shell:

LOAD DATA INPATH 'user/cloudera/stages\_grade/part-00000' INTO TABLE stages\_grade;

LOAD DATA INPATH 'user/cloudera/stages\_grade/part-00001' INTO TABLE stages\_grade;

And then retrieve all the rows in the table with:

SELECT \* FROM stages\_grade;

Output:

stages\_grade.stage stages\_grade.grade

Lowerlevel G-04

Lowerlevel G-04

Lowerlevel G-04

Lowerlevel G-04

Lowerlevel G-04

Lowerlevel G-04

MiddleSchool G-07

MiddleSchool G-07

MiddleSchool G-07

MiddleSchool G-07

MiddleSchool G-07

MiddleSchool G-07

Lowerlevel G-04

.....

2.5.2.2 Create View ‘stagesGradeView’’

Create a view called ‘stagesGradeView’’ that retrieve the stage, minimum level of grade from the ‘stages\_grade’ and ‘grades’ table:

\* ‘grades’ table is already created during the analysis 2.

CREATE VIEW stagesGradeView AS SELECT sg.stage AS stage, MIN(g.intgrade) AS minIntGrade

FROM stages\_grade AS sg

INNER JOIN grades AS g

ON g.grade = sg.grade

GROUP BY sg.stage;

And then retrieve all the rows in the table with:

SELECT \* FROM stagesGradeView;

Output:

Stagesgradeview.stage stagesgradeview.minintgrade

HighSchool 9

MiddleSchool 6

lowerlevel 2

2.5.2.3 Display Analysis Output

Type the following command:

SELECT sgv.stage AS stage, g.grade AS grade

FROM stagesGradeView AS sgv

INNER JOIN grades AS g

ON sgv.minIntGrade = g.intgrade;

Output:

stage grade

HighSchool G-09

MiddleSchool G-06

Lowerlevel G-02

Each educational stage that students score the best grade is shown in the table.

2.6 Analysis 6: Find the nationality of students who has the highest education stage

The last analysis is about finding which nationality students has the most number of students with stageID “HighSchool”. Instead of using Hive queries, we find the highest number of student in Spark. Through sortBy() in descending order, the data with highest number of student will appear at the first row, we can display the row using first() as the result.

2.6.1 Spark

2.6.1.1 Map data into RDD

val highSchool = data.map(x =>(x(1),x(3))).filter(x => x.\_2.contains("HighSchool"))

-map() returns nationality and stageID

-filter() return only row with “HighSchool” as stageID

val countStudent = highSchool.map(x => (x.\_1,1)).reduceByKey(\_+\_)

- map() return the nationality column with an integer 1 for each nationality.

- reduceByKey() group data by nationality and the integer 1 is added up for each nationality.

val highestEduNation = countStudent.sortBy(-\_.\_2).map(x => x.\_1).first()

- sortBy() sort data in descending order according to number of students

- map() returns nationality column

- first() display the first data which is the nationality with the highest number of student

Output:

highestEduNation: String = KW