CCA175 Q&As

CCA Spark and Hadoop Developer Exam

Pass Cloudera CCA175 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.pass2lead.com/cca175.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Cloudera Official Exam Center

QUESTION 1

Problem Scenario 61 : You have been given below code snippet. val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)

val b = a.keyBy(_.length)

val c = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3)

val d = c.keyBy(_.length) operationl

Write a correct code snippet for operationl which will produce desired output, shown below.

Array[(lnt, (String, Option[String]}}] = Array((6,(salmon,Some(salmon))),

(6,(salmon,Some(rabbit))),

(6,(salmon,Some(turkey))), (6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))),

(6,(salmon,Some(turkey))), (3,(dog,Some(dog))), (3,(dog,Some(cat))),

(3,(dog,Some(dog))), (3,(dog,Some(bee))), (3,(rat,Some(dogg)), (3,(rat,Some(cat)j),

(3,(rat.Some(gnu))). (3,(rat,Some(bee))), (8,(elephant,None)))

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution : b.leftOuterJoin(d}.collect leftOuterJoin [Pair]: Performs an left outer join using two key-value RDDs. Please note that the keys must be generally comparable to make this work keyBy : Constructs twocomponent tuples (key-value pairs) by applying a function on each data item. Trie result of the function becomes the key and the original data item becomes the value of the newly created tuples.

QUESTION 2

Problem Scenario 68 : You have given a file as below. spark75/f ile1.txt File contain some text. As given Below spark75/file1.txt Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework The core of Apache Hadoop consists of a storage part known as Hadoop Distributed File System (HDFS) and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. his approach takes advantage of data locality nodes manipulating the data they have access to to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking For a slightly more complicated task, lets look into splitting up sentences from our documents into word bigrams. A bigram is pair of successive tokens in some sequence. We will look at building bigrams from the sequences of words in each sentence, and then try to find the most frequently occuring ones. The first problem is that values in each partition of our initial RDD describe lines from the file rather than sentences. Sentences may be split over multiple lines. The glom() RDD method is used to create a single entry for each document containing the list of all lines, we can then join the lines up, then resplit them into sentences using "." as the separator, using flatMap so that every object in our RDD is now a sentence. A bigram is pair of successive tokens in some sequence. Please build bigrams from the sequences of words in each sentence, and then try to find the most frequently occuring ones.

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

Step 1 : Create all three tiles in hdfs (We will do using Hue}. However, you can first create

in local filesystem and then upload it to hdfs.

Step 2 : The first problem is that values in each partition of our initial RDD describe lines

from the file rather than sentences. Sentences may be split over multiple lines.

The glom() RDD method is used to create a single entry for each document containing the

list of all lines, we can then join the lines up, then resplit them into sentences using "." as

the separator, using flatMap so that every object in our RDD is now a sentence.

sentences = sc.textFile("spark75/file1.txt") \ .glom() \

map(lambda x: " ".join(x)) \ .flatMap(lambda x: x.spllt("."))

Step 3 : Now we have isolated each sentence we can split it into a list of words and extract

the word bigrams from it. Our new RDD contains tuples

containing the word bigram (itself a tuple containing the first and second word) as the first

value and the number 1 as the second value. bigrams = sentences.map(lambda x:x.split())

\ .flatMap(lambda x: [((x[i],x[i+1]),1)for i in range(0,len(x)-1)])

Step 4 : Finally we can apply the same reduceByKey and sort steps that we used in the

wordcount example, to count up the bigrams and sort them in order of descending

frequency. In reduceByKey the key is not an individual word but a bigram.

freq_bigrams = bigrams.reduceByKey(lambda x,y:x+y)\

map(lambda x:(x[1],x[0])) \

sortByKey(False)

freq_bigrams.take(10)

QUESTION 3

Problem Scenario 76 : You have been given MySQL DB with following details. user=retail_dba password=cloudera database=retail_db table=retail_db.orders table=retail_db.order_items jdbc URL = jdbc:mysql://quickstart:3306/retail_db Columns of order table : (orderid , order_date , ordercustomerid, order_status} ..... Please accomplish following activities.

1.

Copy "retail_db.orders" table to hdfs in a directory p91_orders.

2.

Once data is copied to hdfs, using pyspark calculate the number of order for each status.

3.

Use all the following methods to calculate the number of order for each status. (You need to know all these functions and its behavior for real exam)

-countByKey() -groupByKey()

-reduceByKey() -aggregateByKey()

-combineByKey()

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

Step 1 : Import Single table

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db --username=retail dba password=cloudera -table=orders --target-dir=p91_orders

Note : Please check you dont have space between before or after \\'=\\' sign. Sqoop uses the

MapReduce framework to copy data from RDBMS to hdfs

Step 2 : Read the data from one of the partition, created using above command, hadoop fs

-cat p91_orders/part-m-00000

Step 3: countByKey #Number of orders by status allOrders = sc.textFile("p91_orders")

#Generate key and value pairs (key is order status and vale as an empty string keyValue =

aIIOrders.map(lambda line: (line.split(",")[3], ""))

#Using countByKey, aggregate data based on status as a key

output=keyValue.countByKey()Jtems()

for line in output: print(line)

Step 4 : groupByKey

#Generate key and value pairs (key is order status and vale as an one

keyValue = allOrders.map(lambda line: (line.split)",")[3], 1))

#Using countByKey, aggregate data based on status as a key output=

keyValue.groupByKey().map(lambda kv: (kv[0], sum(kv[1]}}}

tor line in output.collect(): print(line}

Step 5 : reduceByKey

#Generate key and value pairs (key is order status and vale as an one

keyValue = allOrders.map(lambda line: (line.split(","}[3], 1))

#Using countByKey, aggregate data based on status as a key output=

keyValue.reduceByKey(lambda a, b: a + b)

tor line in output.collect(): print(line}

Step 6: aggregateByKey

#Generate key and value pairs (key is order status and vale as an one keyValue =

allOrders.map(lambda line: (line.split(",")[3], line}}

output=keyValue.aggregateByKey(0, lambda a, b: a+1, lambda a, b: a+b}

for line in output.collect(): print(line}

Step 7 : combineByKey

#Generate key and value pairs (key is order status and vale as an one

keyValue = allOrders.map(lambda line: (line.split(",")[3], line))

output=keyValue.combineByKey(lambda value: 1, lambda ace, value: acc+1, lambda ace,

value: acc+value)

tor line in output.collect(): print(line)

#Watch Spark Professional Training provided by www.ABCTECH.com to understand more

on each above functions. (These are very important functions for real exam)

QUESTION 4

Problem Scenario 57 : You have been given below code snippet.

val a = sc.parallelize(1 to 9, 3) operationl

Write a correct code snippet for operationl which will produce desired output, shown below.

Array[(String, Seq[lnt])] = Array((even,ArrayBuffer(2, 4, G, 8)), (odd,ArrayBuffer(1, 3, 5, 7,

9)))

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

a.groupBy(x => {if (x % 2 == 0) "even" else "odd" }).collect

QUESTION 5

Problem Scenario 30 : You have been given three csv files in hdfs as below.

EmployeeName.csv with the field (id, name)

EmployeeManager.csv (id, manager Name)

EmployeeSalary.csv (id, Salary)

Using Spark and its API you have to generate a joined output as below and save as a text

tile (Separated by comma) for final distribution and output must be sorted by id.

ld,name,salary,managerName

EmployeeManager.csv

E01,Vishnu

E02,Satyam

E03,Shiv

E04,Sundar

E05,John

E06,Pallavi

E07,Tanvir

E08,Shekhar

E09,Vinod

E10,Jitendra

EmployeeName.csv

E01,Lokesh

E02,Bhupesh

E03,Amit

E04,Ratan E05,Dinesh E06,Pavan E07,Tejas E08,Sheela E09,Kumar E10,Venkat EmployeeSalary.csv E01,50000 E02,50000 E03,45000 E04,45000 E05,50000 E06,45000 E07,50000 E08,10000 E09,10000 E10,10000

Correct Answer: See the explanation for Step by Step Solution and configuration.

Solution :

Step 1 : Create all three files in hdfs in directory called sparkl (We will do using Hue}.

However, you can first create in local filesystem and then

Step 2 : Load EmployeeManager.csv file from hdfs and create PairRDDs

val manager = sc.textFile("spark1/EmployeeManager.csv")

val managerPairRDD = manager.map(x=> (x.split(",")(0),x.split(",")(1)))

Step 3 : Load EmployeeName.csv file from hdfs and create PairRDDs

val name = sc.textFile("spark1/EmployeeName.csv")

val namePairRDD = name.map(x=> (x.split(",")(0),x.split(\\'\")(1)))

Step 4 : Load EmployeeSalary.csv file from hdfs and create PairRDDs

val salary = sc.textFile("spark1/EmployeeSalary.csv")

val salaryPairRDD = salary.map(x=> (x.split(",")(0),x.split(",")(1)))

Step 4 : Join all pairRDDS

val joined = namePairRDD.join(salaryPairRDD}.join(managerPairRDD}

Step 5 : Now sort the joined results, val joinedData = joined.sortByKey()

Step 6 : Now generate comma separated data.

val finalData = joinedData.map(v=> (v._1, v._2._1._1, v._2._1._2, v._2._2))

Step 7 : Save this output in hdfs as text file.

finalData.saveAsTextFile("spark1/result.txt")