CCA175$^{Q\&As}$

CCA Spark and Hadoop Developer Exam

Pass Cloudera CCA175 Exam with 100% Guarantee

QUESTION 1

Problem Scenario 31 : You have given following two files

1.

Content.txt: Contain a huge text file containing space separated words.

2.

Remove.txt: Ignore/filter all the words given in this file (Comma Separated). Write a Spark program which reads the Content.txt file and load as an RDD, remove all the words from a broadcast variables (which is loaded as an RDD of words from Remove.txt). And count the occurrence of the each word and save it as a text file in HDFS. Content.txt Hello this is ABCTech.com This is TechABY.com Apache Spark Training This is Spark Learning Session Spark is faster than MapReduce Remove.txt Hello, is, this, the

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

Solution : Step 1 : Create all three files in hdfs in directory called spark2 (We will do using Hue).However, you can first create in local filesystem and then upload it to hdfs Step 2 : Load the Content.txt file val content = sc.textFile("spark2/Content.txt") //Load the text file Step 3 : Load the Remove.txt file val remove = sc.textFile("spark2/Remove.txt") //Load the text file Step 4 : Create an RDD from remove, However, there is a possibility each word could have trailing spaces, remove those whitespaces as well. We have used two functions here flatMap, map and trim. val removeRDD= remove.flatMap(x=> x.splitf\\',")).map(word=>word.trim)//Create an array of words Step 5 : Broadcast the variable, which you want to ignore val bRemove = sc.broadcast(removeRDD.collect().toList) // It should be array of Strings Step 6 : Split the content RDD, so we can have Array of String. val words = content.flatMap(line => line.split(" ")) Step 7 : Filter the RDD, so it can have only content which are not present in "Broadcast Variable". val filtered = words.filter{case (word) => !bRemove.value.contains(word)} Step 8 : Create a PairRDD, so we can have (word,1) tuple or PairRDD. val pairRDD = filtered.map(word => (word,1)) Step 9 : Nowdo the word count on PairRDD. val wordCount = pairRDD.reduceByKey(_ + _) Step 10 : Save the output as a Text file. wordCount.saveAsTextFile("spark2/result.txt")

QUESTION 2

Problem Scenario 4: You have been given MySQL DB with following details. user=retail_dba password=cloudera database=retail_db table=retail_db.categories jdbc URL = jdbc:mysql://quickstart:3306/retail_db Please accomplish following activities.

Import Single table categories (Subset data} to hive managed table , where category_id between 1 and 22

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

Solution :

Step 1 : Import Single table (Subset data)

sqoop import --connect jdbc:mysql://quickstart:3306/retail_db -username=retail_dba password=cloudera -table=categories -where "\'category_id\' between 1 and 22" --hiveimport

--m 1

Note: Here the \\' is the same you find on ~ key

This command will create a managed table and content will be created in the following

directory.

/user/hive/warehouse/categories

Step 2 : Check whether table is created or not (In Hive)

show tables;

select * from categories;

QUESTION 3

Problem Scenario 69 : Write down a Spark Application using Python, In which it read a file "Content.txt" (On hdfs) with following content. And filter out the word which is less than 2 characters and ignore all empty lines. Once doen store the filtered data in a directory called "problem84" (On hdfs) Content.txt Hello this is ABCTECH.com This is ABYTECH.com Apache Spark TrainingThis is Spark Learning Session Spark is faster than MapReduce

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

Solution : Step 1 : Create an application with following code and store it in problem84.py # Import SparkContext and SparkConf from pyspark import SparkContext, SparkConf # Create configuration object and set App name

conf = SparkConf().setAppName("CCA 175 Problem 84") sc = sparkContext(conf=conf)

#load data from hdfs

contentRDD = sc.textFile(MContent.txt")

#filter out non-empty lines

nonemptyjines = contentRDD.filter(lambda x: len(x) > 0)

#Split line based on space

words = nonempty_lines.ffatMap(lambda x: x.split(\\'\\'}}

#filter out all 2 letter words

finalRDD = words.filter(lambda x: len(x) > 2)

for word in finalRDD.collect():

print(word)

#Save final data finalRDD.saveAsTextFile("problem84M)

step 2 : Submit this application

spark-submit -master yarn problem84.py

QUESTION 4

Problem Scenario 94 : You have to run your Spark application on yarn with each executor

20GB and number of executors should be 50. Please replace XXX, YYY, ZZZ

export HADOOP_CONF_DIR=XXX

./bin/spark-submit \

-class com.hadoopexam.MyTask \

xxx\

-deploy-mode cluster \ # can be client for client mode

YYY\

222 \

/path/to/hadoopexam.jar \

1000

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

Solution

XXX: -master yarn YYY : -executor-memory 20G ZZZ: -num-executors 50

QUESTION 5

Problem Scenario 35 : You have been given a file named spark7/EmployeeName.csv (id,name). EmployeeName.csv E01,Lokesh E02,Bhupesh E03,Amit E04,Ratan E05,Dinesh E06,Pavan E07,Tejas E08,Sheela E09,Kumar E10,Venkat

1. Load this file from hdfs and sort it by name and save it back as (id,name) in results directory. However, make sure while saving it should be able to write In a single file.

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

Solution:

Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local

filesystem and then upload it to hdfs.

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

val name = sc.textFile("spark7/EmployeeName.csv")

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

Step 3 : Now swap namePairRDD RDD.

val swapped = namePairRDD.map(item => item.swap)

step 4: Now sort the rdd by key.

val sortedOutput = swapped.sortByKey()

Step 5 : Now swap the result back

val swappedBack = sortedOutput.map(item => item.swap}

Step 6 : Save the output as a Text file and output must be written in a single file.

swappedBack. repartition(1).saveAsTextFile("spark7/result.txt")