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.passapply.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 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 2

Problem Scenario 12 : You have been given following mysql database details as well as other info. user=retail_dba password=cloudera database=retail_db jdbc URL = jdbc:mysql://quickstart:3306/retail_db Please accomplish following.

1.

Create a table in retailedb with following definition.

CREATE table departments_new (department_id int(11), department_name varchar(45),

created_date T1MESTAMP DEFAULT NOW());

2.

Now isert records from departments table to departments_new

3.

Now import data from departments_new table to hdfs.

4.

Insert following 5 records in departmentsnew table. Insert into departments_new

values(110, "Civil" , null); Insert into departments_new values(111, "Mechanical" , null);

Insert into departments_new values(112, "Automobile" , null); Insert into departments_new

values(113, "Pharma" , null);

Insert into departments_new values(114, "Social Engineering" , null);

5.

Now do the incremental import based on created_date column.

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

Solution :

Step 1 : Login to musql db

mysql --user=retail_dba -password=cloudera

show databases;

use retail db; show tables;

Step 2 : Create a table as given in problem statement.

CREATE table departments_new (department_id int(11), department_name varchar(45),

createddate T1MESTAMP DEFAULT NOW());

show tables;

Step 3 : isert records from departments table to departments_new insert into

departments_new select a.", null from departments a;

Step 4 : Import data from departments new table to hdfs.

sqoop import \

-connect jdbc:mysql://quickstart:330G/retail_db \

~username=retail_dba \

-password=cloudera \

-table departments_new\

--target-dir /user/cloudera/departments_new \

--split-by departments

Stpe 5 : Check the imported data.

hdfs dfs -cat /user/cloudera/departmentsnew/part"

Step 6 : Insert following 5 records in departmentsnew table.

Insert into departments_new values(110, "Civil" , null);

Insert into departments_new values(111, "Mechanical" , null);

Insert into departments_new values(112, "Automobile" , null);

Insert into departments_new values(113, "Pharma" , null);

Insert into departments_new values(114, "Social Engineering" , null);

commit;

Stpe 7 : Import incremetal data based on created_date column.

sqoop import \

-connect jdbc:mysql://quickstart:330G/retaiI_db \

-username=retail_dba \

-password=cloudera \

--table departments_new\

-target-dir /user/cloudera/departments_new \

-append \

-check-column created_date \

-incremental lastmodified \

-split-by departments \

-last-value "2016-01-30 12:07:37.0"

Step 8 : Check the imported value.

hdfs dfs -cat /user/cloudera/departmentsnew/part"

QUESTION 3

Problem Scenario 81 : You have been given MySQL DB with following details. You have been given following product.csv file product.csv productID,productCode,name,quantity,price 1001,PEN,Pen Red,5000,1.23 1002,PEN,Pen Blue,8000,1.25 1003,PEN,Pen Black,2000,1.25 1004,PEC,Pencil 2B,10000,0.48 1005,PEC,Pencil 2H,8000,0.49 1006,PEC,Pencil HB,0,9999.99 Now accomplish following activities.

1.

Create a Hive ORC table using SparkSql

2.

Load this data in Hive table.

3.

Create a Hive parquet table using SparkSQL and load data in it.

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

Solution :

Step 1 : Create this tile in HDFS under following directory (Without header}

/user/cloudera/he/exam/task1/productcsv

Step 2 : Now using Spark-shell read the file as RDD

// load the data into a new RDD

val products = sc.textFile("/user/cloudera/he/exam/task1/product.csv")

// Return the first element in this RDD

prod u cts.fi rst()

Step 3 : Now define the schema using a case class

case class Product(productid: Integer, code: String, name: String, quantity:lnteger, price:

Float)

Step 4 : create an RDD of Product objects

val prdRDD = products.map(_.split(",")).map(p =>

Product(p(0).tolnt,p(1),p(2),p(3}.tolnt,p(4}.toFloat))

prdRDD.first()

prdRDD.count()

Step 5 : Now create data frame val prdDF = prdRDD.toDF()

Step 6 : Now store data in hive warehouse directory. (However, table will not be created }

import org.apache.spark.sql.SaveMode

prdDF.write.mode(SaveMode.Overwrite).format("orc").saveAsTable("product_orc_table")

step 7: Now create table using data stored in warehouse directory. With the help of hive.

hive

show tables

CREATE EXTERNAL TABLE products (productid int,code string,name string .quantity int,

price float}

STORED AS ore

LOCATION 7user/hive/warehouse/product_orc_table\\';

Step 8 : Now create a parquet table

import org.apache.spark.sql.SaveMode

prdDF.write.mode(SaveMode.Overwrite).format("parquet").saveAsTable("product_parquet_

table")

Step 9 : Now create table using this

CREATE EXTERNAL TABLE products_parquet (productid int,code string,name string

.quantity int, price float}

STORED AS parquet

LOCATION 7user/hive/warehouse/product_parquet_table\\';

Step 10 : Check data has been loaded or not.

Select * from products;

Select * from products_parquet;

QUESTION 4

Problem Scenario 95 : You have to run your Spark application on yarn with each executor Maximum heap size to be 512MB and Number of processor cores to allocate on each executor will be 1 and Your main application required three values as input arguments V1 V2 V3. Please replace XXX, YYY, ZZZ ./bin/spark-submit -class com.hadoopexam.MyTask --master yarn-cluster--num-executors 3 --driver-memory 512m XXX YYY lib/hadoopexam.jarZZZ

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

Solution

XXX: -executor-memory 512m YYY: -executor-cores 1 ZZZ : V1 V2 V3 Notes : spark-submit on yarn options Option Description archives Comma-separated list of archives to be extracted into the working directory of each executor. The path must be globally visible inside your cluster; see Advanced Dependency Management. executor-cores Number of processor cores to allocate on each executor. Alternatively, you can use the spark.executor.cores property, executor-memory Maximum heap size to allocate to each executor. Alternatively, you can use the spark.executor.memory-property. num-executors Total number of YARN containers to allocate for this application. Alternatively, you can use the spark.executor.instances property. queue YARN queue to submit to. For more information, see Assigning Applications and Queries to Resource Pools. Default: default.

QUESTION 5

Problem Scenario 27 : You need to implement near real time solutions for collecting information when submitted in file with below information.

Data

echo "IBM,100,20160104" >> /tmp/spooldir/bb/.bb.txt echo "IBM,103,20160105" >> /tmp/spooldir/bb/.bb.txt mv /tmp/spooldir/bb/.bb.txt /tmp/spooldir/bb/bb.txt After few mins echo "IBM,100.2,20160104" >> /tmp/spooldir/dr/.dr.txt echo "IBM,103.1,20160105" >> /tmp/spooldir/dr/.dr.txt mv /tmp/spooldir/dr/.dr.txt /tmp/spooldir/dr/dr.txt

Requirements:

You have been given below directory location (if not available than create it) /tmp/spooldir .

You have a finacial subscription for getting stock prices from BloomBerg as well as

Reuters and using ftp you download every hour new files from their respective ftp site in

directories /tmp/spooldir/bb and /tmp/spooldir/dr respectively.

As soon as file committed in this directory that needs to be available in hdfs in

/tmp/flume/finance location in a single directory.

Write a flume configuration file named flume7.conf and use it to load data in hdfs with

following additional properties .

1.

Spool /tmp/spooldir/bb and /tmp/spooldir/dr

2.

File prefix in hdfs sholuld be events

3.

File suffix should be .log

4.

If file is not commited and in use than it should have _ as prefix.

5.

Data should be written as text to hdfs

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

Solution : Step 1 : Create directory mkdir /tmp/spooldir/bb mkdir /tmp/spooldir/dr Step 2 : Create flume configuration file, with below configuration for agent1.sources = source1 source2 agent1 .sinks = sink1 agent1.channels = channel1 agent1 .sources.source1.channels = channel1 agentl .sources.source2.channels = channell agent1 .sinks.sinkl.channel = channell agent1 .sources.source1.type = spooldir agent1 .sources.sourcel.spoolDir = /tmp/spooldir/bb agent1 .sources.source2.type = spooldir agent1 .sources.source2.spoolDir = /tmp/spooldir/dr agent1 .sinks.sink1.type = hdfs agent1 .sinks.sink1.hdfs.path = /tmp/flume/finance agent1-sinks.sink1.hdfs.filePrefix = events agent1.sinks.sink1.hdfs.fileSuffix = .log agent1 .sinks.sink1.hdfs.inUsePrefix = _ agent1 .sinks.sink1.hdfs.fileType = Data Stream agent1.channels.channel1.type = file Step 4 : Run below command which will use this configuration file and append data in hdfs. Start flume service: flume-ng agent -conf /home/cloudera/flumeconf -conf-file /home/cloudera/flumeconf/flume7.conf --name agent1 Step 5 : Open another terminal and create a file in /tmp/spooldir/ echo "IBM,100,20160104" » /tmp/spooldir/bb/.bb.txt echo "IBM,103,20160105" » /tmp/spooldir/bb/.bb.txt mv /tmp/spooldir/bb/.bb.txt /tmp/spooldir/bb/bb.txt After few mins echo "IBM,100.2,20160104" » /tmp/spooldir/dr/.dr.txt echo "IBM,103.1,20160105" »/tmp/spooldir/dr/.dr.txt mv /tmp/spooldir/dr/.dr.txt /tmp/spooldir/dr/dr.txt