**HU Extension Assignment 05 E63 Big Data Analytics**

Issued on: Sept 29, 2017 Due by 4 PM EST, Oct 07, 2017

Implement solution for this assignment on Cloudera Quick Start VM with CDH 5.12.

**Problem 1.** Download Quick Start VM for CDH 5.12 from <https://www.cloudera.com/downloads/quickstart_vms/5-8.html>. Start the VM. Please assign to the VM as much memory as you can. Examine whether hadoop-hdfs-\* , hadoop-mapreduce-\* and hadoop-yarn-\* daemons are running. If those daemons are not running start all of them. If any of daemons fails to run, try to fix it.

[10%]

|  |
| --- |
| **[[cloudera@quickstart conf.pseudo]$ for x in `cd /etc/init.d ; ls hadoop-\*`; do sudo service $x status ; done**  Hadoop datanode is running [ OK ]  Hadoop journalnode is running [ OK ]  Hadoop namenode is running [ OK ]  Hadoop secondarynamenode is running [ OK ]  Hadoop httpfs is running [ OK ]  Hadoop historyserver is running [ OK ]  Hadoop nodemanager is running [ OK ]  Hadoop proxyserver is dead and pid file exists [ OK ]  Hadoop resourcemanager is running [ OK ] |

**Problem 2.** Examine whether there are HDFS directories for users: spark, hive, oozie, and cloudera. If the directories are present, find the content of those directories. If the directories are not present, create them. Please do not format the namenode.

[10%]

|  |
| --- |
| **[cloudera@quickstart init.d]$ hadoop fs -ls /user**  Found 8 items  drwxr-xr-x - cloudera cloudera 0 2017-07-19 06:28 /user/cloudera  drwxr-xr-x - mapred hadoop 0 2017-07-19 06:29 /user/history  drwxrwxrwx - hive supergroup 0 2017-07-19 06:31 /user/hive  drwxrwxrwx - hue supergroup 0 2017-07-19 06:30 /user/hue  drwxrwxrwx - jenkins supergroup 0 2017-07-19 06:29 /user/jenkins  drwxrwxrwx - oozie supergroup 0 2017-07-19 06:30 /user/oozie  drwxrwxrwx - root supergroup 0 2017-07-19 06:29 /user/root  drwxr-xr-x - hdfs supergroup 0 2017-07-19 06:31 /user/spark |

|  |
| --- |
| **[cloudera@quickstart init.d]$ hadoop fs -ls /user/spark**  Found 1 items  drwxrwxrwx - spark supergroup 0 2017-10-02 12:18 /user/spark/applicationHistory  **[cloudera@quickstart init.d]$ hadoop fs -ls /user/hive**  Found 1 items  drwxrwxrwx - hive supergroup 0 2017-07-19 06:31 /user/hive/warehouse  **[cloudera@quickstart init.d]$ hadoop fs -ls /user/oozie**  Found 1 items  drwxrwxrwx - oozie supergroup 0 2017-07-19 06:30 /user/oozie/share  **[cloudera@quickstart init.d]$ hadoop fs -ls /user/cloudera** |

**Problem 3**. Create new Linux user smith. Make that user a member of the mapredLinux group. Make that user a sudo user. Create the home directory of user smith in HDFS. Download provided files bible.tar and shakespeare.tar. Unzip both tar files and copy the resulting files into HDFS directory input of user smith. As user smith run Hadoop grep on both bible and shakespeare texts. Every Hadoop run requires separate output directory. Examine content of first 20 lines of files generated by Hadoop grep.

[15%]

user: smith

password: cloudera

|  |
| --- |
| **hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep home/cloudera/Desktop/all-bible /home/smith/bible\_freq '\w+'** |

**Problem 4**. Create your own version of “Hadoop grep” program using Spark. Compare your results with the results of Hadoop grep when applied to the texts of King James Bible, and all of Shakespeare’s works, contained in files bible.tar and shakespear.tar respectively. Notice small differences between results obtained by your Spark program and Hadoop grep. Try to explain what causes those differences. Save results of your Spark grep operations both in HDFS and on your local file system. You can implement your solution using one of interactive shells or a standalone program.

[20%]

**LOCAL FILE SYSTEM**

|  |
| --- |
| **BIBLE**  **ul = sc.textFile("file:////home/cloudera/Desktop/all-bible")**  **counts = ul.flatMap(lambda x:x.split(" ")).map(lambda x:**  **(x,1)).reduceByKey(add)**  **print(counts.take(5))**  **exchanged = counts.map(lambda x: (x[1],x[0]))**  **print(exchanged.take(5))**  **sorted = exchanged.sortBy(lambda x: x[0], ascending=False)**  **print(sorted.take(20))**  [(605968, u''), (62384, u'the'), (38711, u'and'), (34618, u'of'), (13505, u'to'), (12735, u'And'), (12478, u'that'), (12279, u'in'), (9764, u'shall'), (9513, u'he'), (8930, u'unto'), (8708, u'I'), (8362, u'his'), (8054, u'a'), (7183, u'for'), (6897, u'they'), (6754, u'be'), (6747, u'is'), (6047, u'with'), (5878, u'not')]  **Shakespeare**  **shake\_ul = sc.textFile("file:////home/cloudera/Desktop/all-shakespeare")**  **shake\_counts = shake\_ul.flatMap(lambda x:x.split(" ")).map(lambda x:**  **(x,1)).reduceByKey(add)**  **print(shake\_counts.take(5))**  **shake\_exchanged = shake\_counts.map(lambda x: (x[1],x[0]))**  **print(shake\_exchanged.take(5))**  **shake\_sorted = shake\_exchanged.sortBy(lambda x: x[0], ascending=False)**  **print(shake\_sorted.take(20))**  [(64531, u''), (25069, u'the'), (18793, u'and'), (16436, u'to'), (16069, u'of'), (15223, u'I'), (12982, u'a'), (11180, u'my'), (10134, u'in'), (9109, u'you'), (8109, u'is'), (7773, u'that'), (7123, u'not'), (7001, u'with'), (6594, u'his'), (6202, u'be'), (6119, u'your'), (5955, u'\tAnd'), (5781, u'for'), (5311, u'have')] |

**Problem 5**. Create your own tables KINGJAMES with columns for words and frequencies and insert into the table the result of your Spark grep program which produces word counts in file bible. Find all words in table KINGJAMES which start with letter “w” and are 4 or more characters long and appear more than 250 times. Write a query that will tell us the number of such words. Before counting turn all words in lower case.

When comparing a word with a string your use LIKE operator, like

word like ‘a%’ or word like ‘%th%’

Symbol ‘%’ means any number of characters. You measure the length of a string using function length() and you change the case of a word to all lower characters using function lower().

[20%]

|  |
| --- |
| bible\_lines = sc.textFile("file:////home/cloudera/Desktop/all-bible")\  .flatMap(lambda l: l.split())\  .map(lambda x: re.sub("[^a-zA-Z]+", "", x.lower().encode("utf-8", "ignore"))) \  .filter(lambda x: x != "")  print(bible\_lines.take(10))  bible\_words = bible\_lines.map(lambda p: Row( bible\_word=str(p) ) )  print(bible\_words.take(10))  bible\_df = sqlContext.createDataFrame(bible\_words)  print(bible\_df.show(10))  bible\_df.registerTempTable("KINGJAMES")  bible = sqlContext.sql("""  SELECT  bible\_word,  COUNT(\*) freq  FROM KINGJAMES  WHERE lower(bible\_word) like 'w%'  AND length(bible\_word) > 4  GROUP BY bible\_word  HAVING COUNT(\*) > 250  """)  bible = bible.orderBy(bible['freq'].desc())  print(bible.show()) |

**OUTPUT**

|  |
| --- |
| ['king', 'james', 'bible', 'body', 'backgroundfaebd', 'margin', 'textalignjustify', 'p', 'textindent', 'em']  [Row(bible\_word='king'), Row(bible\_word='james'), Row(bible\_word='bible'), Row(bible\_word='body'), Row(bible\_word='backgroundfaebd'), Row(bible\_word='margin'), Row(bible\_word='textalignjustify'), Row(bible\_word='p'), Row(bible\_word='textindent'), Row(bible\_word='em')]  +----------------+  | bible\_word|  +----------------+  | king|  | james|  | bible|  | body|  | backgroundfaebd|  | margin|  |textalignjustify|  | p|  | textindent|  | em|  +----------------+  only showing top 10 rows  None  +----------+----+  |bible\_word|freq|  +----------+----+  | which|4427|  | words| 548|  | would| 451|  | without| 442|  | where| 407|  | water| 396|  | woman| 357|  | wherefore| 348|  | wicked| 344|  | whose| 314|  |wilderness| 304|  | works| 302|  | world| 287|  | waters| 287|  | written| 283|  +----------+----+ |

**Problem 6**. Transfer content of your Hive KINGJAMES table to a Spark DataFrame. Perform the analysis from problem 6 using any available API in Spark. Please note that you are working with Spark 1.6.

[20%]

|  |
| --- |
|  |

**Problem 7.** Use Sqoop to transfer the content of MySQL database retail\_db which is present on the Cloudera VM into Hive. Demonstrate that new Hive tables are created and correspond to the original MySQL tables. Find the number of rows in each table. Compare those row counts with row counts in MySQL database.

[15%]

|  |
| --- |
|  |

We are also attaching two groups of example data files for Hive: examples\_older.zip and hive\_examples.zip. You might find those files useful if you want to keep on learning about the technology. You could get those files by downloading Hive distributions, as described in notes.

.