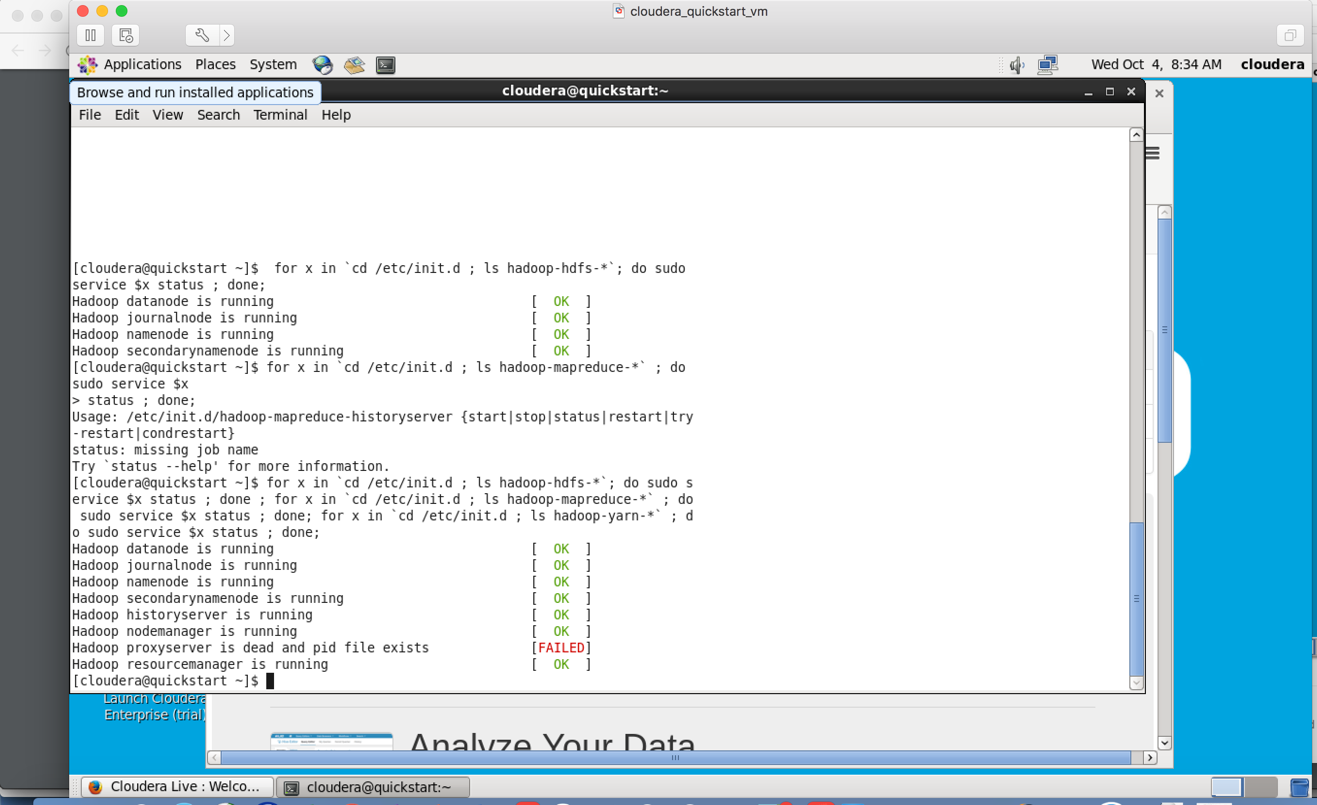
**HU Extension Assignment 05 E63 Big Data Analytics**

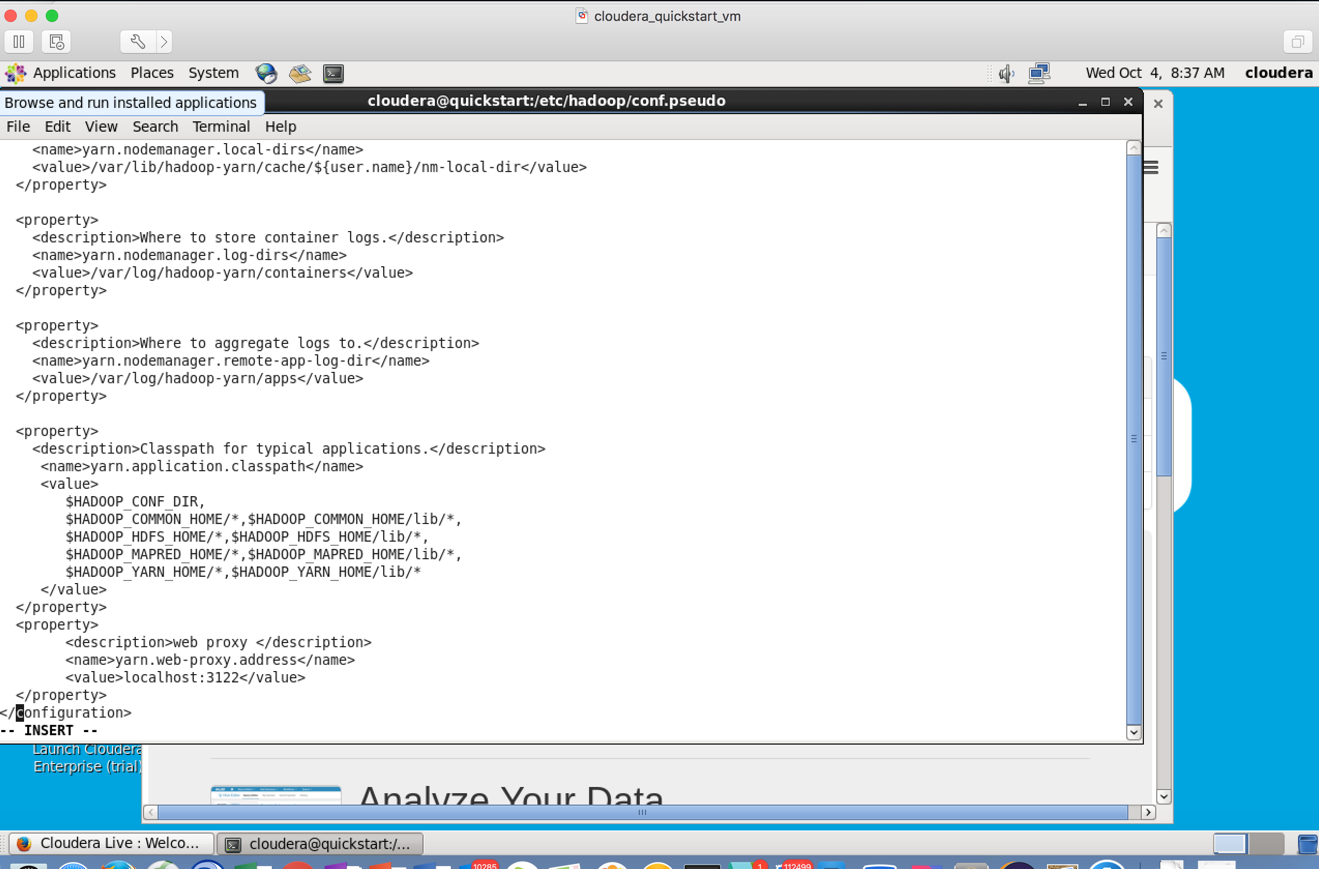
Issued on: Sept 29, 2017 Due by 4 PM EST, Oct 07, 2017

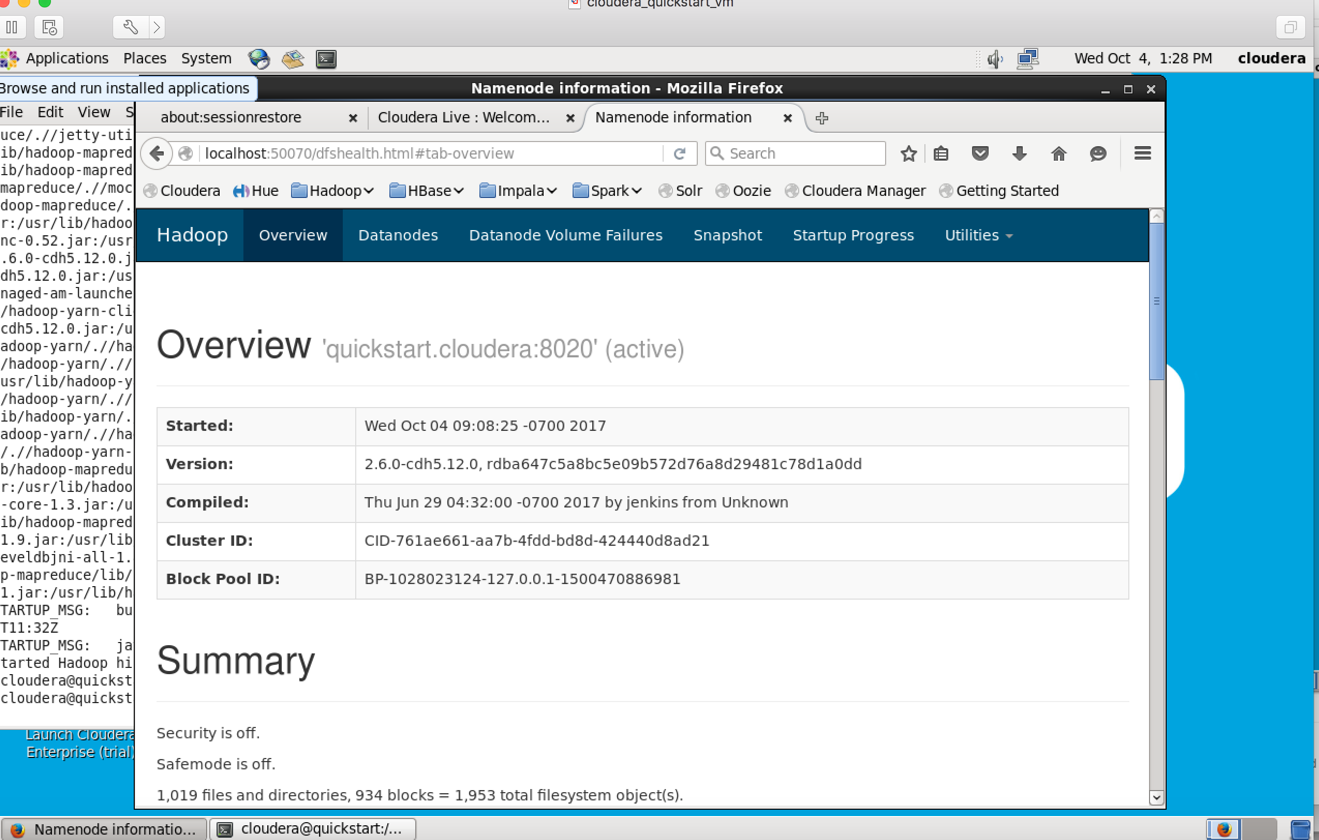
Implement solution for this assignment on Cloudera Quick Start VM with CDH 5.12.

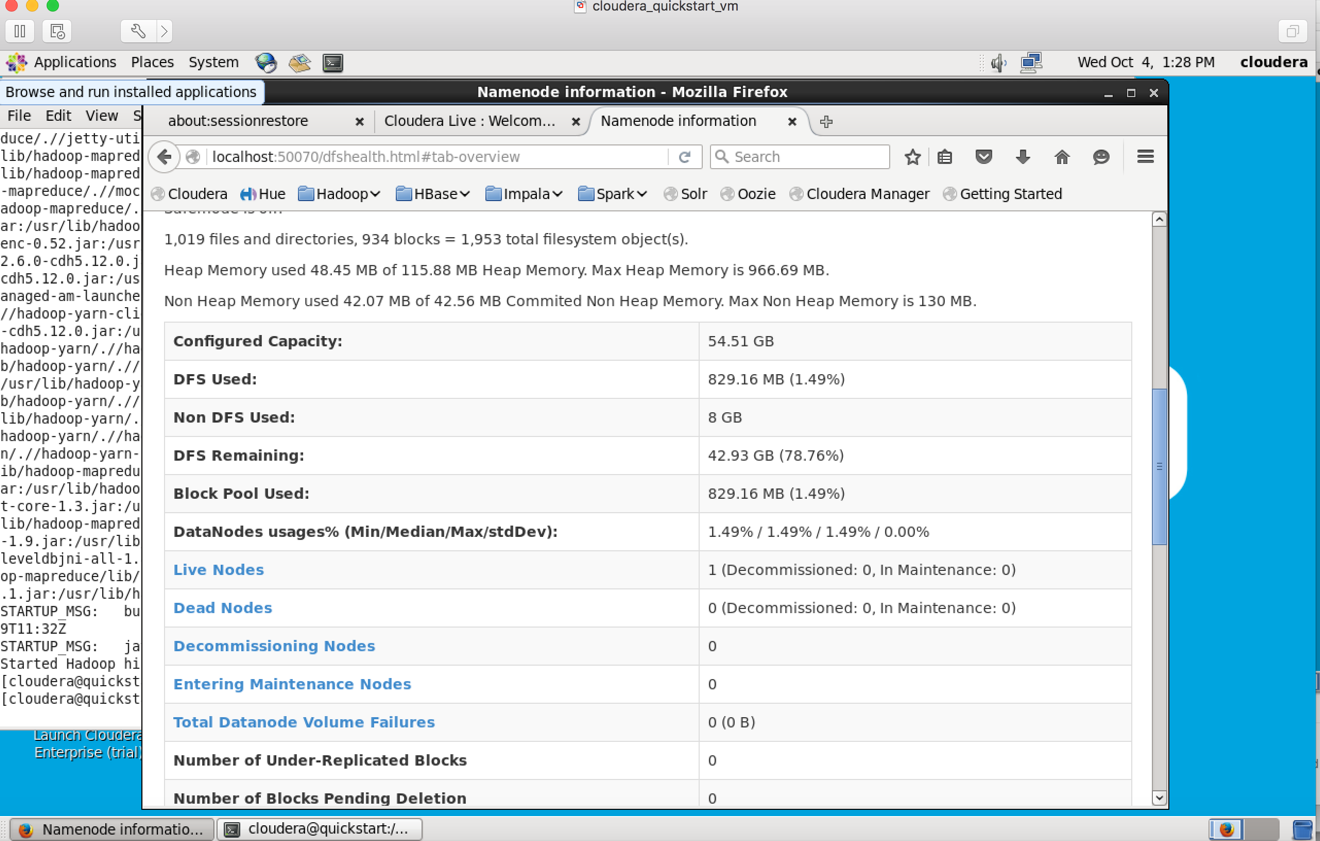
**Problem 1.** Download Quick Start VM for CDH 5.12 from <https://www.cloudera.com/downloads/quickstart_vms/5-8.html>. Start the VM. Please assign to the VM as much memory as you can. Examine whether hadoop-hdfs-\* , hadoop-mapreduce-\* and hadoop-yarn-\* daemons are running. If those daemons are not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

[10%]



****

****

****

**[cloudera@quickstart ~]$ for x in `cd /etc/init.d ; ls hadoop-hdfs-\*`; do sudo service $x status ; done ; for x in `cd /etc/init.d ; ls hadoop-mapreduce-\*` ; do sudo service $x status ; done; for x in `cd /etc/init.d ; ls hadoop-yarn-\*` ; 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 historyserver is running [ OK ]

Hadoop nodemanager is running [ OK ]

Hadoop proxyserver is running [ OK ]

Hadoop resourcemanager is running [ OK ]

[cloudera@quickstart ~]$

//Faced some issues here with the history server not starting up right.Got errors like this //when I checked /var/log/hadoop-mapreduce/mapred-mapred-historyserver-//quickstart.cloudera.out.5

17/10/04 09:25:27 WARN ipc.Client: Failed to connect to server: quickstart.cloudera/192.168.168.129:8020: try once and fail.

java.net.NoRouteToHostException: No route to host

at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)

at sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:739)

at org.apache.hadoop.net.SocketIOWithTimeout.connect(SocketIOWithTimeout.java:206)

at org.apache.hadoop.net.NetUtils.connect(NetUtils.java:530)

at org.apache.hadoop.net.NetUtils.connect(NetUtils.java:494)

at org.apache.hadoop.ipc.Cl

Updated /etc/hosts and did a restart like this:

**sudo /etc/init.d/hadoop-mapreduce-historyserver restart**

**Problem 2.** Examine whether there are HDFS home 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 ~]$ hadoop fs -ls /user/

Found 9 items

drwxr-xr-x - cloudera cloudera 0 2017-10-05 01:36 /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

Went inside these directories and checked their content.

**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%]

//Create user, add to group mapred

[root@quickstart ~]# **useradd -g mapred smith**

[root@quickstart ~]# **passwd smith**

New password:

Retype new password:

passwd: all authentication tokens updated successfully.

//Ref: <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/2/html/Getting_Started_Guide/ch02s03.html>

//Sudo access

visudo

uncomment

**%wheel ALL=(ALL) NOPASSWD: ALL**

**usermod -aG wheel smith**

[smith@quickstart root]$ **groups**

mapred wheel

[smith@quickstart root]$ **sudo whoami**

root

//Create Hadoop home dir

[smith@quickstart root]$ **sudo -u hdfs hadoop fs -mkdir /user/smith**

**hadoop fs -ls /user/**

Found 9 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 2017-10-04 13:44 /user/smith

drwxr-xr-x - hdfs supergroup 0 2017-07-19 06:31 /user/spark

//Untarred

ls -lrt

total 20612

-rw-r--r-- 1 smith mapred 5284231 Nov 29 2012 all-shakespeare

-rw-r--r-- 1 smith mapred 5258688 Nov 29 2012 all-bible

-rw-r--r-- 1 smith mapred 5263360 Oct 4 14:03 bible.tar

-rw-r--r-- 1 smith mapred 5294080 Oct 4 14:03 shakespeare.tar

[smith@quickstart ~]$ head all-shakespeare

ALL'S WELL THAT ENDS WELL

DRAMATIS PERSONAE

KING OF FRANCE (KING:)

//Created hdfs dir input

[smith@quickstart ~]$ **hadoop fs -mkdir /user/smith/input**

[smith@quickstart ~]$ **hadoop fs -put all\* /user/smith/input/**

[smith@quickstart ~]$ **hadoop fs -ls /user/smith/input**

Found 2 items

-rw-r--r-- 1 smith supergroup 5258688 2017-10-04 14:06 /user/smith/input/all-bible

-rw-r--r-- 1 smith supergroup 5284231 2017-10-04 14:06 /user/smith/input/all-shakespeare

//Run the hadoop grep for bible text

**hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep input/all-bible bible\_freq '\w+'**

17/10/04 14:31:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1507133341661\_0003

17/10/04 14:31:08 INFO mapred.YARNRunner: Job jar is not present. Not adding any jar to the list of resources.

17/10/04 14:31:08 INFO impl.YarnClientImpl: Submitted application application\_1507133341661\_0003

17/10/04 14:31:08 INFO mapreduce.Job: The url to track the job: http://localhost:3122/proxy/application\_1507133341661\_0003/

17/10/04 14:31:08 INFO mapreduce.Job: Running job: job\_1507133341661\_0003

17/10/04 14:31:15 INFO mapreduce.Job: Job job\_1507133341661\_0003 running in uber mode : false

17/10/04 14:31:15 INFO mapreduce.Job: map 0% reduce 0%

17/10/04 14:31:23 INFO mapreduce.Job: map 100% reduce 0%

17/10/04 14:31:32 INFO mapreduce.Job: map 100% reduce 100%

17/10/04 14:31:32 INFO mapreduce.Job: Job job\_1507133341661\_0003 completed successfully

17/10/04 14:31:32 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=256987

FILE: Number of bytes written=763997

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=346588

HDFS: Number of bytes written=147408

HDFS: Number of read operations=7

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=1

Launched reduce tasks=1

Data-local map tasks=1

Total time spent by all maps in occupied slots (ms)=5349

Total time spent by all reduces in occupied slots (ms)=5482

Total time spent by all map tasks (ms)=5349

Total time spent by all reduce tasks (ms)=5482

Total vcore-milliseconds taken by all map tasks=5349

Total vcore-milliseconds taken by all reduce tasks=5482

Total megabyte-milliseconds taken by all map tasks=5477376

Total megabyte-milliseconds taken by all reduce tasks=5613568

Map-Reduce Framework

Map input records=14330

Map output records=14330

Map output bytes=228321

Map output materialized bytes=256987

Input split bytes=141

Combine input records=0

Combine output records=0

Reduce input groups=614

Reduce shuffle bytes=256987

Reduce input records=14330

Reduce output records=14330

Spilled Records=28660

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=155

CPU time spent (ms)=2990

Physical memory (bytes) snapshot=405401600

Virtual memory (bytes) snapshot=3015712768

Total committed heap usage (bytes)=347869184

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=346447

File Output Format Counters

Bytes Written=147408

//Now do a head

[smith@quickstart ~]$ **hadoop fs -text /user/smith/bible\_freq/part-r-00000 | head -20**

62394 the

38985 and

34654 of

13526 to

12846 And

12603 that

12445 in

9764 shall

9672 he

8940 unto

8854 I

8385 his

8057 a

7270 for

6974 they

6913 be

6884 is

6649 him

6647 LORD

6591 not

//Do the same for Shakespeare text

[smith@quickstart ~]$  **hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep input/all-shakespeare shakespeare\_freq '\w+'**

17/10/04 14:37:36 INFO mapred.YARNRunner: Job jar is not present. Not adding any jar to the list of resources.

17/10/04 14:37:36 INFO impl.YarnClientImpl: Submitted application application\_1507133341661\_0005

17/10/04 14:37:36 INFO mapreduce.Job: The url to track the job: http://localhost:3122/proxy/application\_1507133341661\_0005/

17/10/04 14:37:36 INFO mapreduce.Job: Running job: job\_1507133341661\_0005

17/10/04 14:37:45 INFO mapreduce.Job: Job job\_1507133341661\_0005 running in uber mode : false

17/10/04 14:37:45 INFO mapreduce.Job: map 0% reduce 0%

17/10/04 14:38:04 INFO mapreduce.Job: map 67% reduce 0%

17/10/04 14:38:05 INFO mapreduce.Job: map 100% reduce 0%

17/10/04 14:38:15 INFO mapreduce.Job: map 100% reduce 100%

17/10/04 14:38:15 INFO mapreduce.Job: Job job\_1507133341661\_0005 completed successfully

17/10/04 14:38:15 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=525117

FILE: Number of bytes written=1300269

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=707396

HDFS: Number of bytes written=299379

HDFS: Number of read operations=7

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=1

Launched reduce tasks=1

Data-local map tasks=1

Total time spent by all maps in occupied slots (ms)=18201

Total time spent by all reduces in occupied slots (ms)=6373

Total time spent by all map tasks (ms)=18201

Total time spent by all reduce tasks (ms)=6373

Total vcore-milliseconds taken by all map tasks=18201

Total vcore-milliseconds taken by all reduce tasks=6373

Total megabyte-milliseconds taken by all map tasks=18637824

Total megabyte-milliseconds taken by all reduce tasks=6525952

Map-Reduce Framework

Map input records=29183

Map output records=29183

Map output bytes=466745

Map output materialized bytes=525117

Input split bytes=141

Combine input records=0

Combine output records=0

Reduce input groups=631

Reduce shuffle bytes=525117

Reduce input records=29183

Reduce output records=29183

Spilled Records=58366

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=287

CPU time spent (ms)=6850

Physical memory (bytes) snapshot=379654144

Virtual memory (bytes) snapshot=3015426048

Total committed heap usage (bytes)=347869184

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=707255

File Output Format Counters

Bytes Written=299379

//do a head of the output for Shakespeare text

**hadoop fs -text /user/smith/shakespeare\_freq/part-r-00000 | head -20**

25578 the

23027 I

19654 and

17462 to

16444 of

13524 a

12697 you

11296 my

10699 in

8857 is

8851 that

8402 not

8033 me

8020 s

7800 And

7231 with

7165 it

6812 his

6753 be

6246 your

**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%]

//Added these to bash\_profile

export JAVA\_HOME=/usr/java/jdk1.7.0\_67-cloudera export SPARK\_HOME=/usr/lib/spark export HADOOP\_MAPRED\_HOME=/usr/lib/hadoop-mapreduce export PATH=$PATH:$JAVA\_HOME/bin:$SPARK\_HOME/bin

//Started spark master

cd /usr/lib/spark/sbin

sudo ./start-master.sh

//started spark-shell

//Do the steps for the bible text file

**val inputBible = sc.textFile("/user/smith/input/all-bible")**

inputBible: org.apache.spark.rdd.RDD[String] = /user/smith/input/all-bible MapPartitionsRDD[1] at textFile at <console>:27

**//**do transformations on RDD to get comma separated output

**val inputBibleCount = inputBible.flatMap(x=>x.split("\\W+")).map(\_.trim).map(\_.replaceAll("[,.!?;\*^~%:\"]()","")).filter(\_.nonEmpty).map((\_,1)).reduceByKey(\_+\_).sortBy(\_.\_2,false).map{case(x,y) => Array(x,y).mkString(",")}**

//write to hadoop

**inputBibleCount.saveAsTextFile("hdfs:///home/cloudera/outputBibleSpark")**

//Check output

[cloudera@quickstart init.d]$ **hadoop fs -text /home/cloudera/outputBibleSpark/part-00000 | head -20**

the,62394

and,38985

of,34654

to,13526

And,12846

that,12603

in,12445

shall,9764

he,9672

unto,8940

I,8854

his,8385

a,8057

for,7270

they,6974

be,6913

is,6884

him,6649

LORD,6647

not,6591

//write to disk

**inputBibleCount.saveAsTextFile("file:////home/cloudera/outputBible")**

//Similarly for Shakespeare text

scala> **val inputShakespeare = sc.textFile("/user/smith/input/all-shakespeare")**

inputShakespeare: org.apache.spark.rdd.RDD[String] = /user/smith/input/all-shakespeare MapPartitionsRDD[119] at textFile at <console>:27

//create rdd, sort by freq desc

scala> **val inputShakespeareCount = inputShakespeare.flatMap(x=>x.split("\\W+")).map(\_.trim).map(\_.replaceAll("[,.!?;\*^~%:\"]()","")).filter(\_.nonEmpty).map((\_,1)).reduceByKey(\_+\_).sortBy(\_.\_2,false)**

inputShakespeareCount: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[128] at sortBy at <console>:29

scala>**inputShakespeareCount.saveAsTextFile("hdfs:///home/cloudera/outputShakespeareSpark")**

//see how the out put looks like

[cloudera@quickstart init.d]$ **hadoop fs -text /home/cloudera/outputShakespeareSpark/part-00000 | head**

(the,25578)

(I,23027)

(and,19654)

(to,17462)

(of,16444)

(a,13524)

(you,12697)

(my,11296)

(in,10699)

(is,8857)

**inputShakespeareCount.saveAsTextFile("file:////home/cloudera/outputShakespeareSpark")**

**ls /home/cloudera/\*output\***

/home/cloudera/outputShakespeareBible:

part-00000 \_SUCCESS

/home/cloudera/outputShakespeareSpark:

part-00000 \_SUCCESS

**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().

[15%]

//Create kingjames table in hive

hive>

**create table KINGJAMES (words String,freq INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile;**

//Load data from output from question before

**LOAD DATA INPATH "file:///home/cloudera/outputBible/part-00000" into table KINGJAMES;**

//Validate

hive> **select \* from kingjames limit 10;**

OK

the 62394

and 38985

of 34654

to 13526

And 12846

that 12603

in 12445

shall 9764

he 9672

unto 8940

//words starting with w, with length 4 or more,where their occurrence is more than 250. I use a inline query for it.

hive> **select \* from (select sum(freq) totalFreq,lower(words) lowerwords from kingjames where lower(words) like 'w%' and length(words) > 3 group by lower(words)) a where totalFreq > 250;**

Total MapReduce CPU Time Spent: 4 seconds 540 msec

OK

396 water

287 waters

264 well

1400 went

2772 were

987 what

2836 when

407 where

348 wherefore

4427 which

765 whom

314 whose

344 wicked

407 wife

304 wilderness

3850 will

6110 with

442 without

364 woman

701 word

546 words

514 work

301 works

288 world

451 would

283 written

//To get count

**select count(1) from (select sum(freq) totalFreq,lower(words) lowerwords from kingjames where lower(words) like 'w%' and length(words) > 3 group by lower(words)) a where totalFreq > 250;**

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.96 sec HDFS Read: 155011 HDFS Write: 114 SUCCESS

Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 2.32 sec HDFS Read: 4778 HDFS Write: 3 SUCCESS

Total MapReduce CPU Time Spent: 6 seconds 280 msec

OK

**26**

**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.

[15%]

//Copy the hive-site.xml

**sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/**

**sudo cp /etc/hive/conf/hive-site.xml /etc/spark/conf/**

//create hive context

scala> **val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)**

hiveContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveContext@2acd7ebd

scala> **val hive\_tables=hiveContext.sql("show tables").foreach(println)**

[**kingjames**,false]

hive\_tables: Unit = ()

//Create the Dataframe from the table

scala> **val kingJames=hiveContext.sql("select \* from kingjames").toDF();**

//Register this as table to query from

**kingJames.registerTempTable("kingJames")**

//Query using sql

scala**> val result = hiveContext.sql("select \* from (select sum(freq) totalFreq,lower(words) lowerwords from kingjames where lower(words) like 'w%' and length(words) > 3 group by lower(words)) a where totalFreq > 250")**

result: org.apache.spark.sql.DataFrame = [totalFreq: bigint, lowerwords: string]

//Check output

scala> **result.show(100,false)**

+---------+----------+

|totalFreq|lowerwords|

+---------+----------+

|2836 |when |

|451 |would |

|1400 |went |

|442 |without |

|6110 |with |

|283 |written |

|4427 |which |

|348 |wherefore |

|407 |where |

|987 |what |

|546 |words |

|301 |works |

|765 |whom |

|2772 |were |

|314 |whose |

|288 |world |

|701 |word |

|287 |waters |

|514 |work |

|304 |wilderness|

|264 |well |

|396 |water |

|3850 |will |

|364 |woman |

|407 |wife |

|344 |wicked |

scala> **result.count**

res5: Long = **26**

This count matches

**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%]

//login to mysql

mysql -uretail\_dba -p

mysql> **show databases;**

+--------------------+

| Database |

+--------------------+

| information\_schema |

| retail\_db |

+--------------------+

//login to retail\_db schema

mysql> **use retail\_db**

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

//Running the import here

**sqoop import-all-tables -m 1 --connect jdbc:mysql://quickstart:3306/retail\_db -username=retail\_dba -password=cloudera -compression-codec=snappy -as-parquetfile -warehouse-dir=/user/hive/warehouse -hive-import**

17/10/05 01:40:53 INFO mapreduce.Job: Counters: 30

File System Counters

FILE: Number of bytes read=0

FILE: Number of bytes written=225840

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

….

….

….

….

17/10/05 01:40:53 INFO mapreduce.ImportJobBase: Transferred 46.1328 KB in 36.9522 seconds (1.2484 KB/sec)

17/10/05 01:40:53 INFO mapreduce.ImportJobBase: Retrieved 1345 records.

//after import see all tables in hive:

hive> **show tables;**

OK

categories

customers

departments

kingjames

order\_items

orders

products

//in mysql

mysql> show tables;

+---------------------+

| Tables\_in\_retail\_db |

+---------------------+

| categories |

| customers |

| departments |

| order\_items |

| orders |

| products |

+---------------------+

//Taking a count in mysql

mysql> **select count(1) from categories;**

+----------+

| count(1) |

+----------+

| 58 |

+----------+

1 row in set (0.00 sec)

mysql> **select count(1) from customers;**

+----------+

| count(1) |

+----------+

| 12435 |

+----------+

1 row in set (0.04 sec)

mysql> **select count(1) from departments;**

+----------+

| count(1) |

+----------+

| 6 |

+----------+

1 row in set (0.00 sec)

mysql> **select count(1) from order\_items;**

+----------+

| count(1) |

+----------+

| 172198 |

+----------+

1 row in set (0.10 sec)

mysql> **select count(1) from orders;**

+----------+

| count(1) |

+----------+

| 68883 |

+----------+

1 row in set (0.04 sec)

mysql> **select count(1) from products;**

+----------+

| count(1) |

+----------+

| 1345 |

+----------+

1 row in set (0.00 sec)

//Now repeating the same in hive

**select count(1) from categories;**

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.7 sec HDFS Read: 10053 HDFS Write: 3 SUCCESS

Total MapReduce CPU Time Spent: 2 seconds 700 msec

OK

**58**

**select count(1) from customers;**

MapReduce Total cumulative CPU time: 3 seconds 220 msec

Ended Job = job\_1507133341661\_0025

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.22 sec HDFS Read: 264693 HDFS Write: 6 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 220 msec

OK

**12435**

**select count(1) from departments;**

MapReduce Total cumulative CPU time: 2 seconds 670 msec

Ended Job = job\_1507133341661\_0026

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.67 sec HDFS Read: 8569 HDFS Write: 2 SUCCESS

Total MapReduce CPU Time Spent: 2 seconds 670 msec

OK

6

**select count(1) from order\_items;**

MapReduce Total cumulative CPU time: 3 seconds 460 msec

Ended Job = job\_1507133341661\_0027

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.46 sec HDFS Read: 1656399 HDFS Write: 7 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 460 msec

OK

**172198**

**select count(1) from orders;**

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 3.26 sec HDFS Read: 496528 HDFS Write: 6 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 260 msec

OK

**68883**

**select count(1) from products;**

apReduce Total cumulative CPU time: 2 seconds 740 msec

Ended Job = job\_1507133341661\_0029

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.74 sec HDFS Read: 53987 HDFS Write: 5 SUCCESS

Total MapReduce CPU Time Spent: 2 seconds 740 msec

OK

1345

//All the counts between hive and mysql match up.

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.

.