CMPE 282 – HW3

SJSU ID: 010034700

Student Name: Keertikeya Gupta

Q1.

Ans:

High level explanation of MapReduce Job:

There are two MapReduce jobs. The first job retrieves the hit count for all the URIs in the three weblog files and stores the information as key-value pair, where key is the URI and value is the hit count. Since the Mapper is using URIs as the key and hit count as value, the output of this job is sorted according to the URIs

The second job takes the above output file as its input, with key as value and value as key. This way, the sorting is done by the original value, i.e. the hit count of the URIs. The reducer then writes the output with URIs as key and hit count as value.

Input directory on VM:

/user/cloudera/input

Output directory on VM:

1. /user/cloudera/count_out (this has the result of first job)
2. /user/cloudera/sorted_out (this has the result of the second job)

# of map tasks:

The first job has one map task and uses three splits. The second job also has one map task, and uses one split. Total there are two map tasks.

# of reduce tasks:

Both first and second jobs have one reducer task each. Total two reducer tasks.

Q2.

Ans.

In the implementation, we have two MapReduce jobs. The first MapReduce job takes the three weblog files as input, parses through them, and produces an output containing all the URIs in the three input files and their respective hit counts. In the mapper class, we split the input by using StringTokenizer. Next, we traverse through the list that is given by StringTokenizer and see whether or not the current value in the list is a URI. If it is, we write it in the mapper's output, else we simply ignore it and move to the next value. The hit count value written for each URI match is 1.

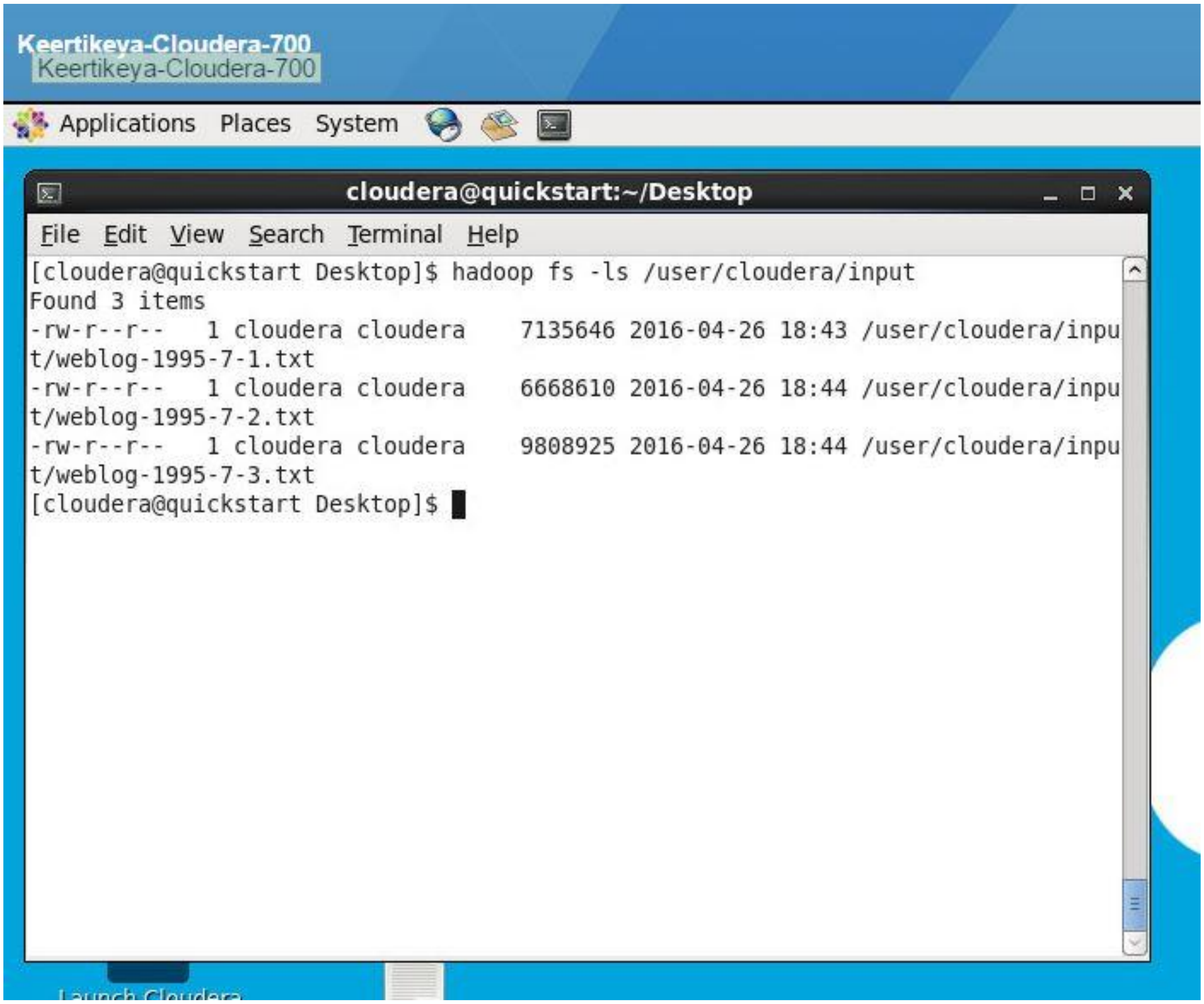
The reducer of this job takes the mapper's output as its input and aggregates the hit count. This gives us the total hit count of all URIs. The URIs are then written into the output file of the reducer task with URIs and their respective total hit counts.

In the second MapReduce job, we take the output file of the first MapReduce job as input. The mapper takes the key-value pair of this input and reads it as value-key. This allows us to sort the data according to the hit counts. Then, the reducer takes the output of the second mapper and writes the value-key as key-value, thus giving us the original format of the data (i.e. URI Hit-count), but sorted according to the hit-counts instead of the URIs.

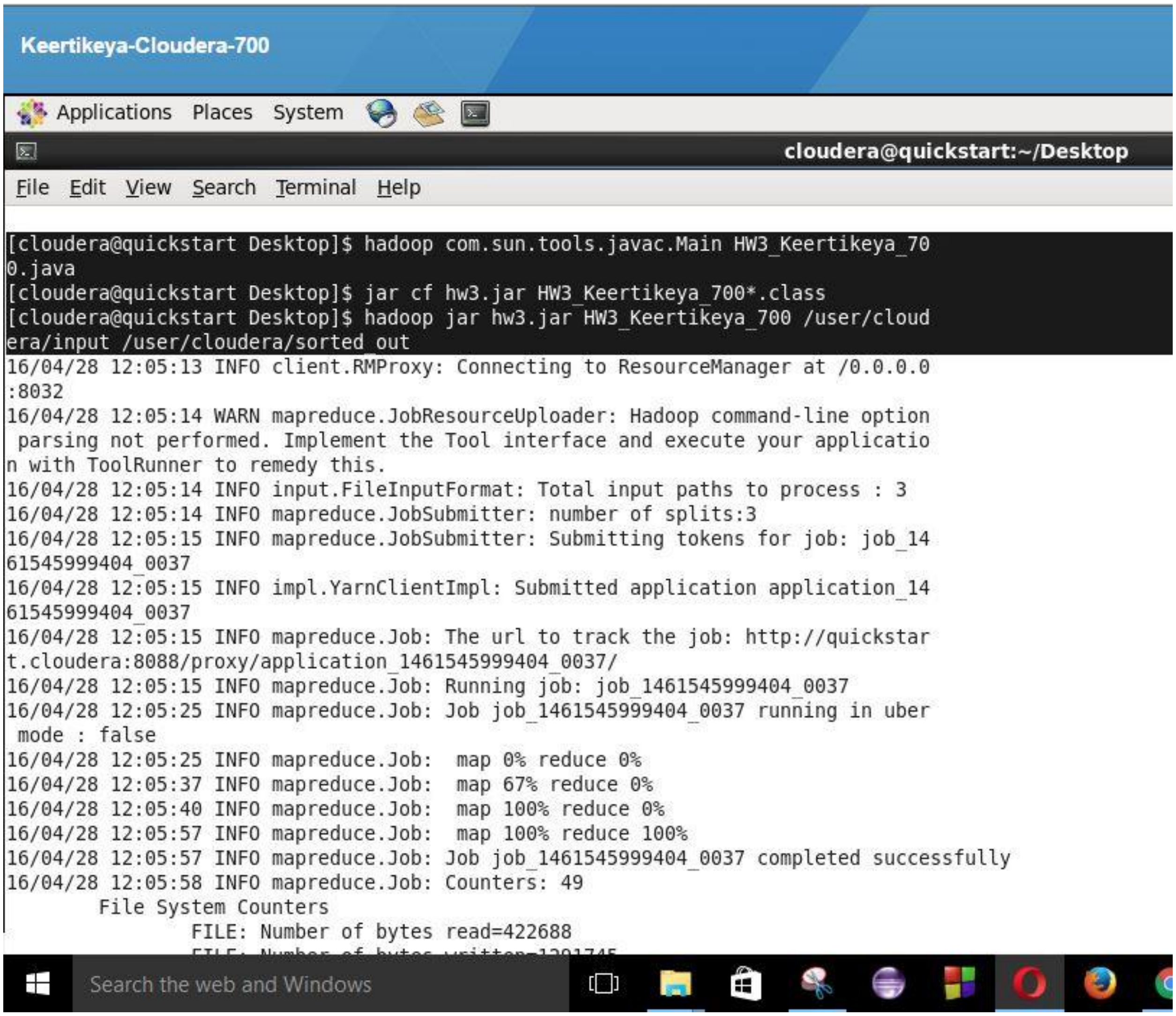
Q3.

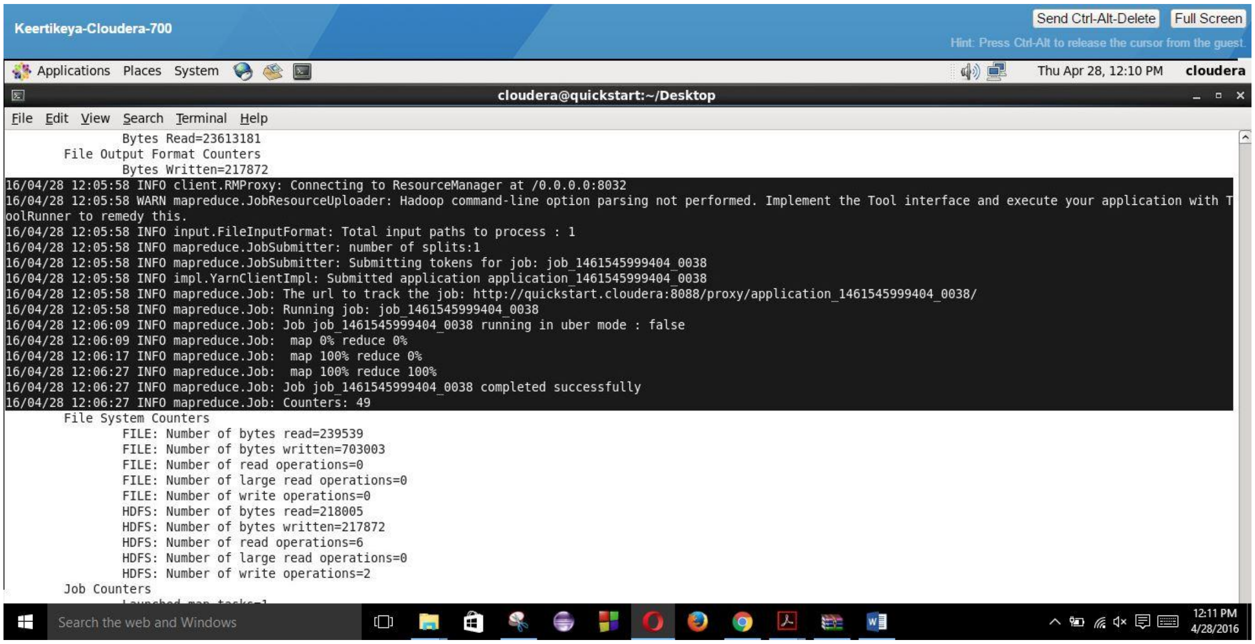
Ans

hadoop fs –ls /user/cloudera/input

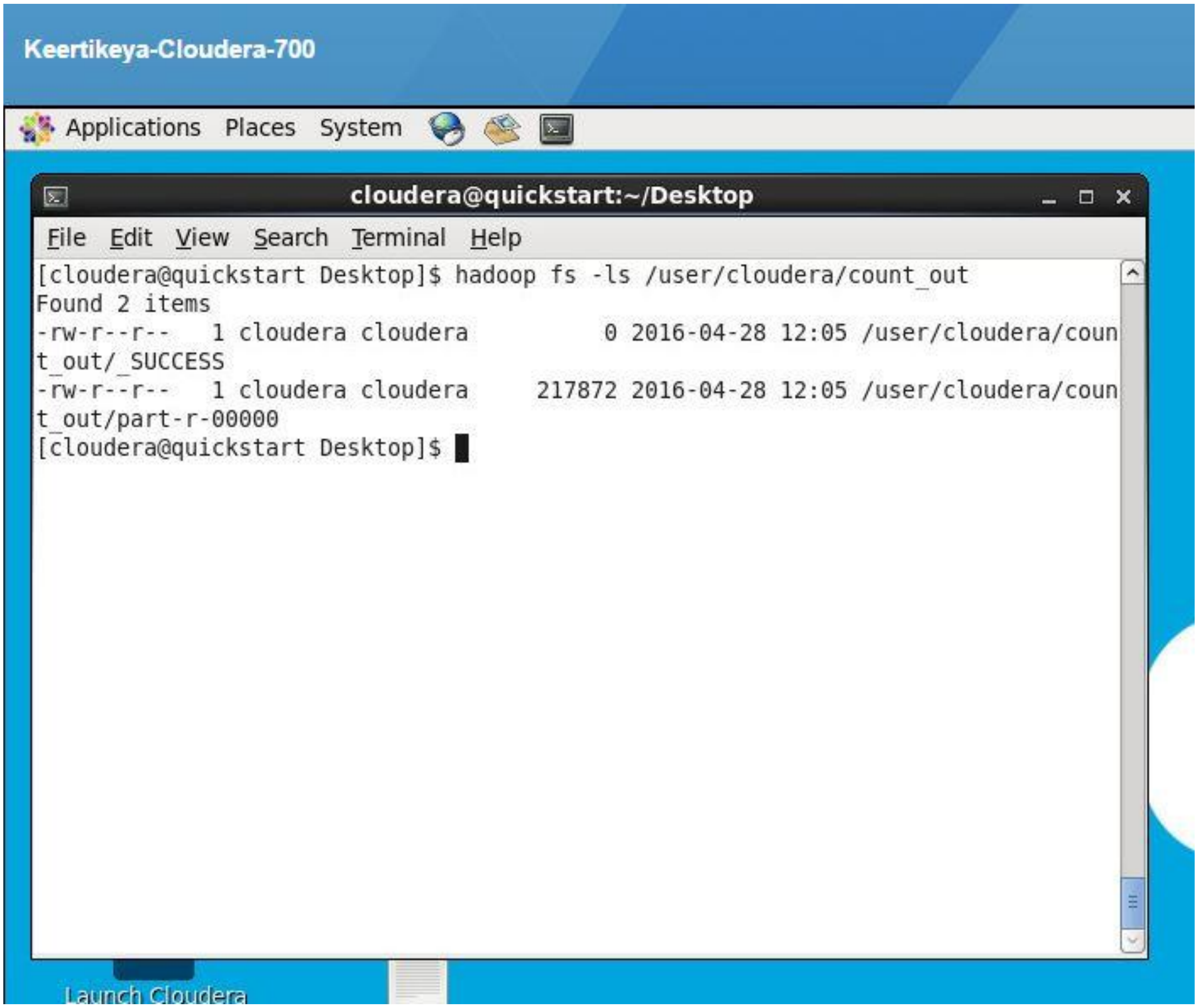


hadoop jar …

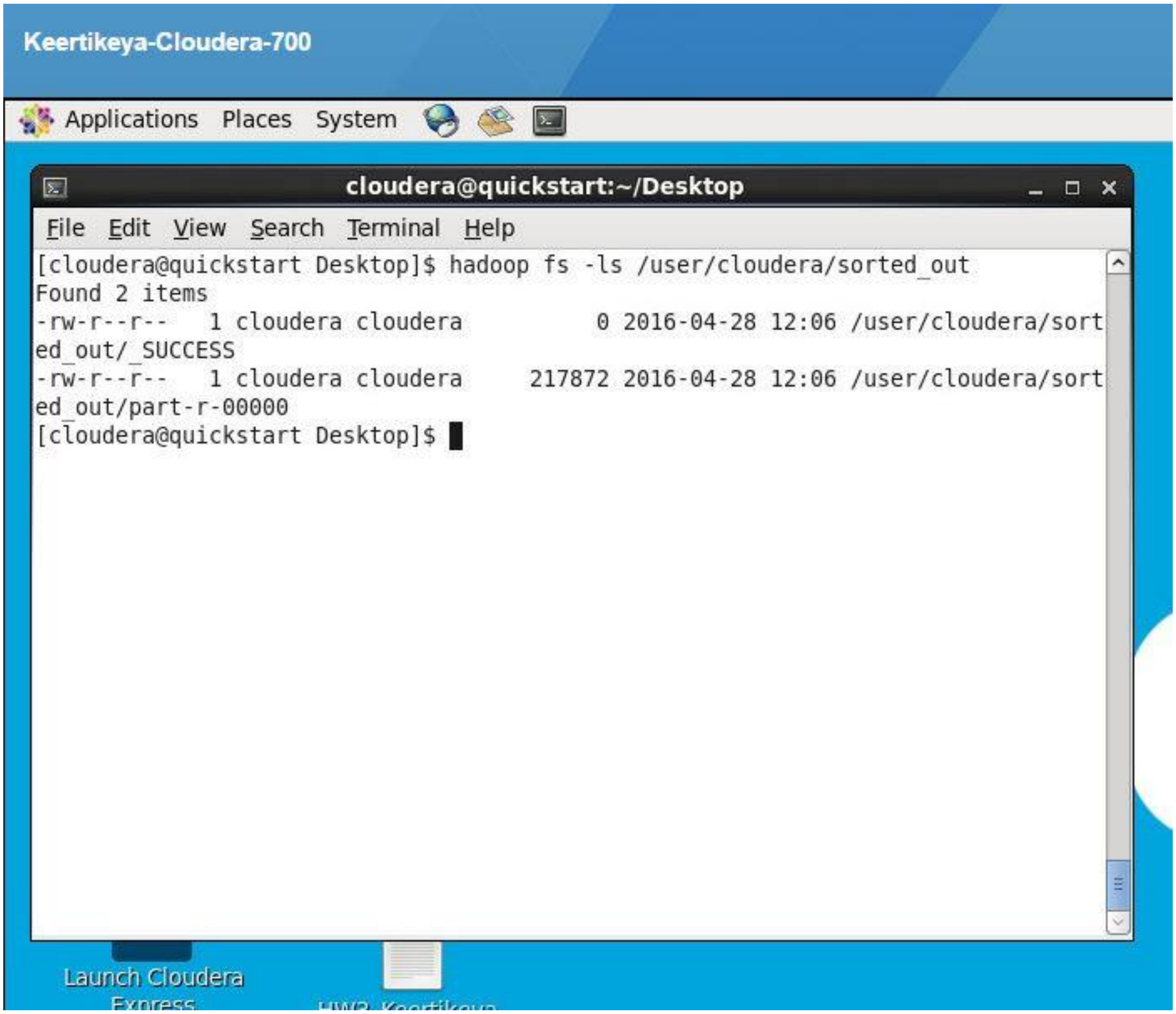

Keertikeya-Cloudera-700
Applications Places System
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
[cloudera@quickstart Desktop]$ hadoop com.sun.tools.javac.Main HW3_Keertikeya_700.java
[cloudera@quickstart Desktop]$ jar cf hw3.jar HW3_Keertikeya_700*.class
[cloudera@quickstart Desktop]$ hadoop jar hw3.jar HW3_Keertikeya_700 /user/cloudera/input /user/cloudera/sorted out
16/04/28 12:05:13 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/04/28 12:05:14 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
16/04/28 12:05:14 INFO input.FileInputFormat: Total input paths to process : 3
16/04/28 12:05:14 INFO mapreduce.JobSubmitter: number of splits:3
16/04/28 12:05:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1461545999404_0037
16/04/28 12:05:15 INFO impl.YarnClientImpl: Submitted application application_1461545999404_0037
16/04/28 12:05:15 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1461545999404_0037/
16/04/28 12:05:15 INFO mapreduce.Job: Running job: job_1461545999404_0037
16/04/28 12:05:25 INFO mapreduce.Job: Job job_1461545999404_0037 running in uber mode : false
16/04/28 12:05:25 INFO mapreduce.Job: map 0% reduce 0%
16/04/28 12:05:37 INFO mapreduce.Job: map 67% reduce 0%
16/04/28 12:05:40 INFO mapreduce.Job: map 100% reduce 0%
16/04/28 12:05:57 INFO mapreduce.Job: map 100% reduce 100%
16/04/28 12:05:57 INFO mapreduce.Job: Job job_1461545999404_0037 completed successfully
16/04/28 12:05:58 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=422688
Search the web and Windows


Keertikeya-Cloudera-700
Send Ctrl-Alt-Delete
Full Screen
Hint: Press Ctrl-Alt to release the cursor from the guest.
Applications Places System
Thu Apr 28, 12:10 PM
cloudera
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
Bytes Read=23613181
File Output Format Counters
Bytes Written=217872
16/04/28 12:05:58 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/04/28 12:05:58 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with T
oolRunner to remedy this.
16/04/28 12:05:58 INFO input.FileInputFormat: Total input paths to process : 1
16/04/28 12:05:58 INFO mapreduce.JobSubmitter: number of splits:1
16/04/28 12:05:58 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1461545999404_0038
16/04/28 12:05:58 INFO impl.YarnClientImpl: Submitted application application_1461545999404_0038
16/04/28 12:05:58 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1461545999404_0038/
16/04/28 12:05:58 INFO mapreduce.Job: Running job: job_1461545999404_0038
16/04/28 12:06:09 INFO mapreduce.Job: Job job_1461545999404_0038 running in uber mode : false
16/04/28 12:06:09 INFO mapreduce.Job: map 0% reduce 0%
16/04/28 12:06:17 INFO mapreduce.Job: map 100% reduce 0%
16/04/28 12:06:27 INFO mapreduce.Job: map 100% reduce 100%
16/04/28 12:06:27 INFO mapreduce.Job: Job job_1461545999404_0038 completed successfully
16/04/28 12:06:27 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=239539
FILE: Number of bytes written=703003
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=218005
HDFS: Number of bytes written=217872
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Search the web and Windows
12:11 PM
4/28/2016

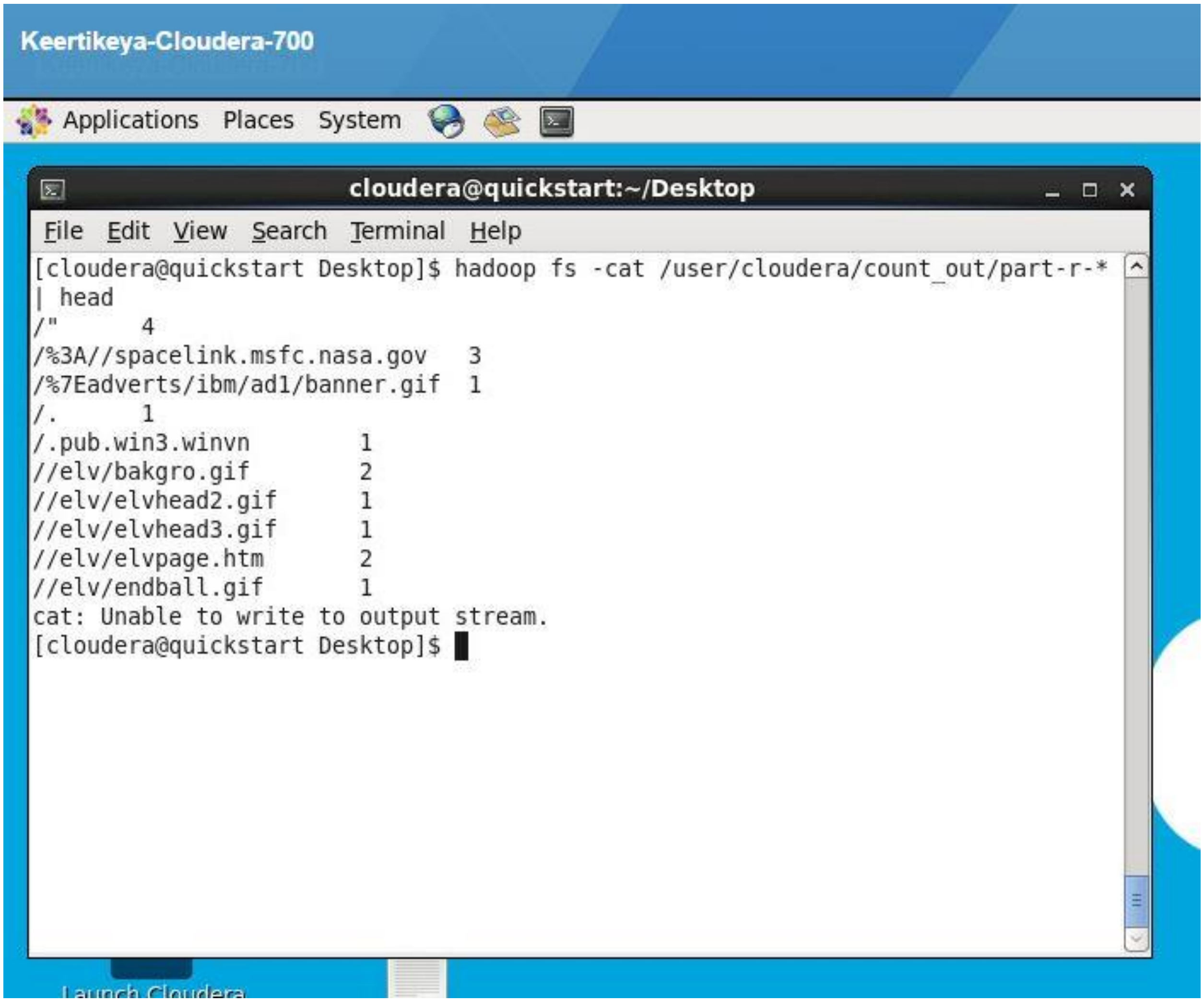
hadoop fs –ls /user/cloudera/count_out **(output of first MapReduce job)**



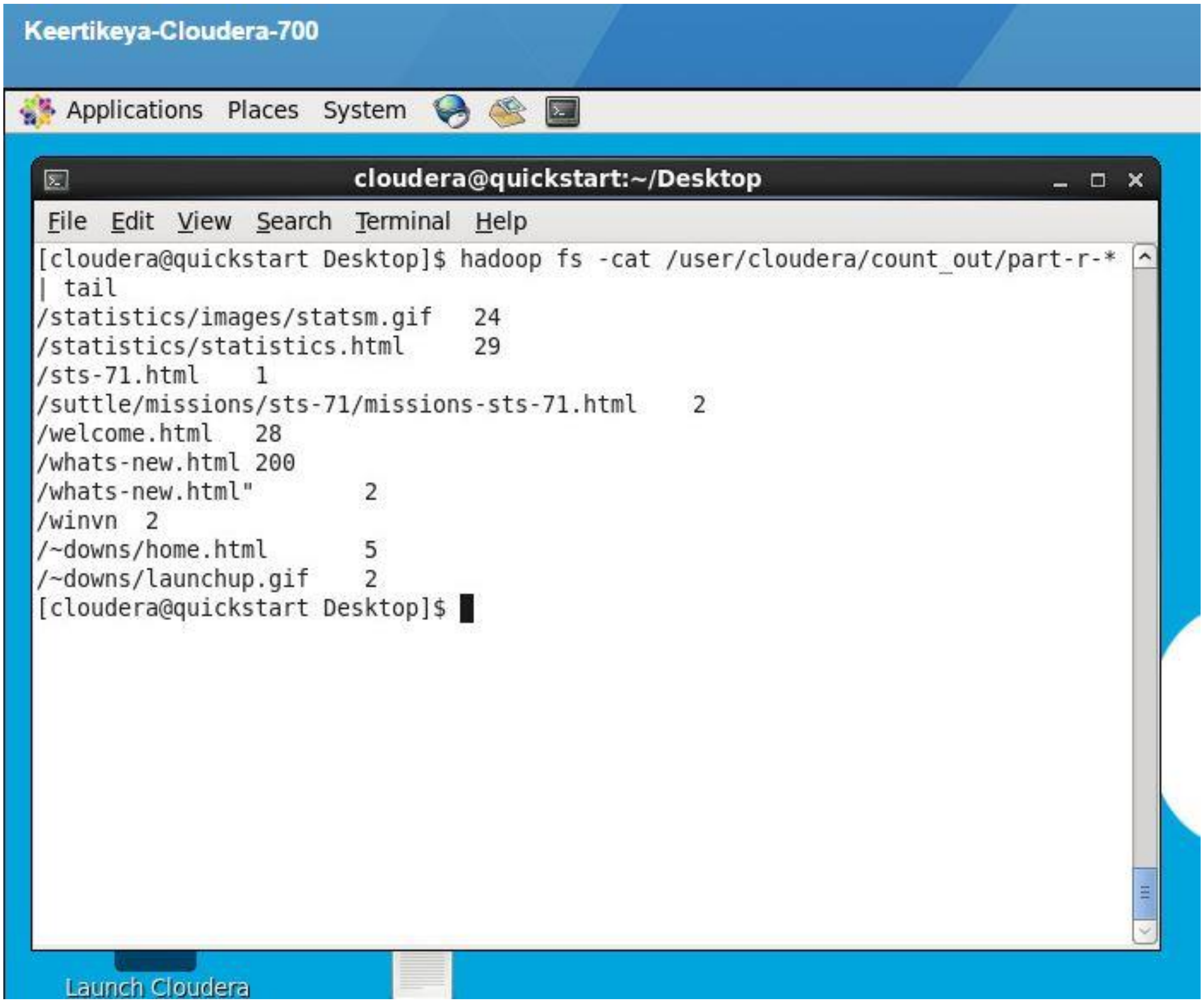
hadoop fs –ls /user/cloudera/sorted_out   (final output directory)



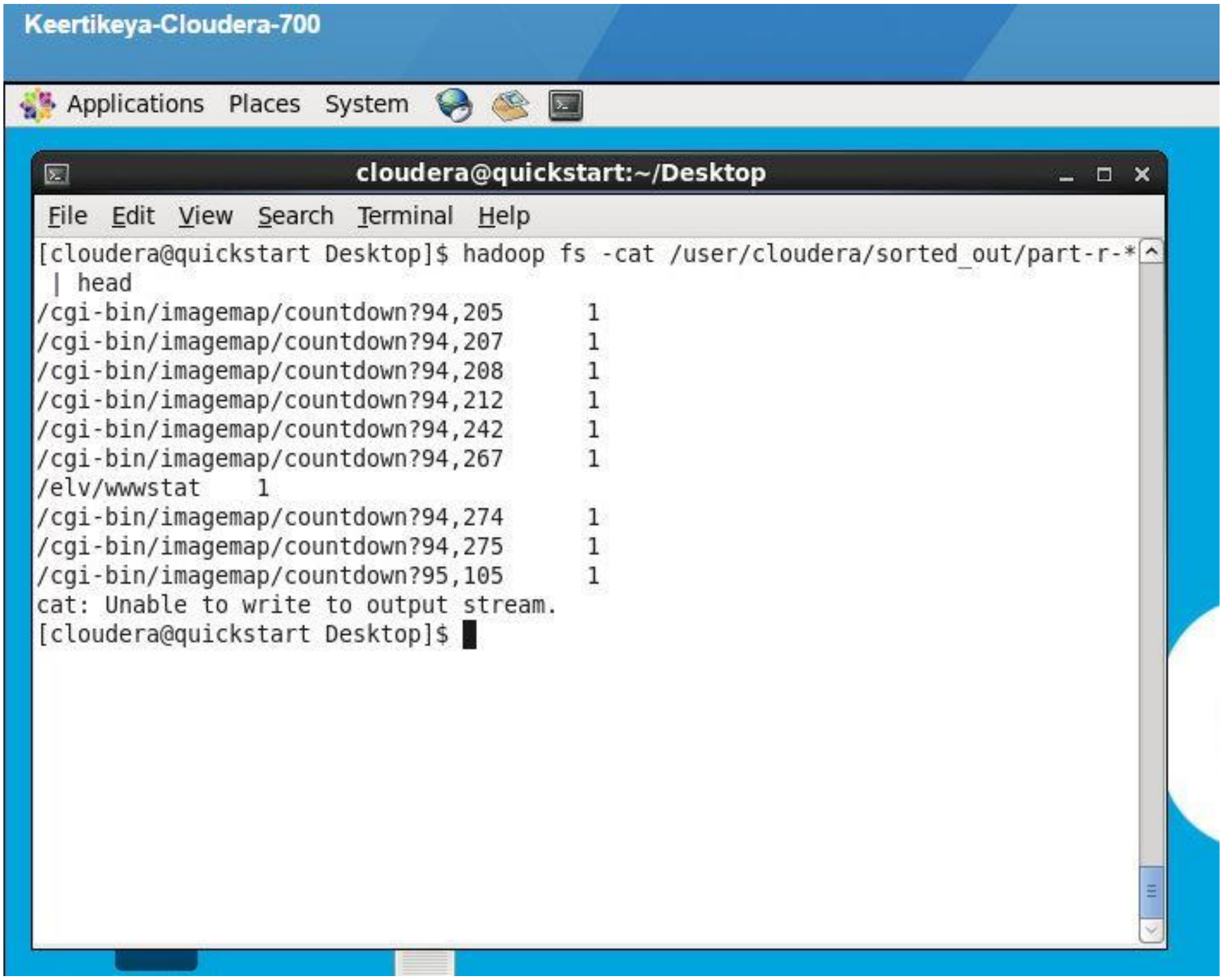
hadoop fs –cat /user/cloudera/count_out/part-r-* | head (first MapReduce job output)



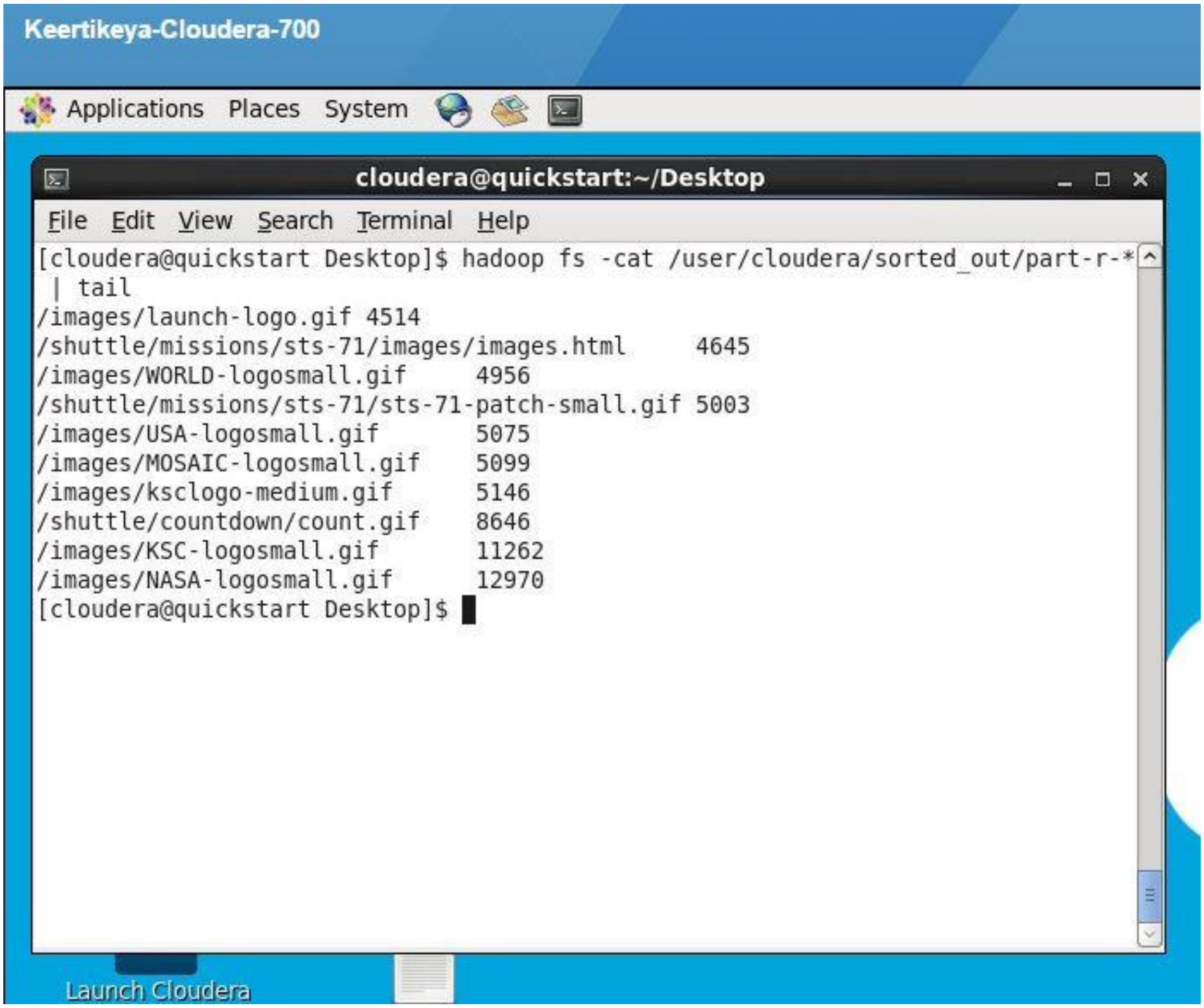
hadoop fs –cat /user/cloudera/count_out/part-r-* | tail (first MapReduce job output)



hadoop fs –cat /user/cloudera/sorted_out/part-r-* | head (final MapReduce job output)



hadoop fs –cat /user/cloudera/sorted_out/part-r-* | tail (final MapReduce job output)



Q4.

Ans.

The performance of the 2nd MapReduce job is better if we use inbuilt partition. Basically there are two ways in which we can do the sorting by value:

First is that we use in-memory sorting. This has better performance since it is in-memory and does not require additional operations. However, the drawback is that if the input file is too big, then the MapReduce job may run out of memory and the operation wouldn't perform at all. In other words, the scalability is poor.

The second method is to use Secondary sorting, in which we create composite key, create our own partitioner and comparator. This has the advantage that we do not risk running out of memory, but the drawback is that the performance is poor compared to the previous option.

Hence, there is always a tradeoff between performance and scalability.