**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.

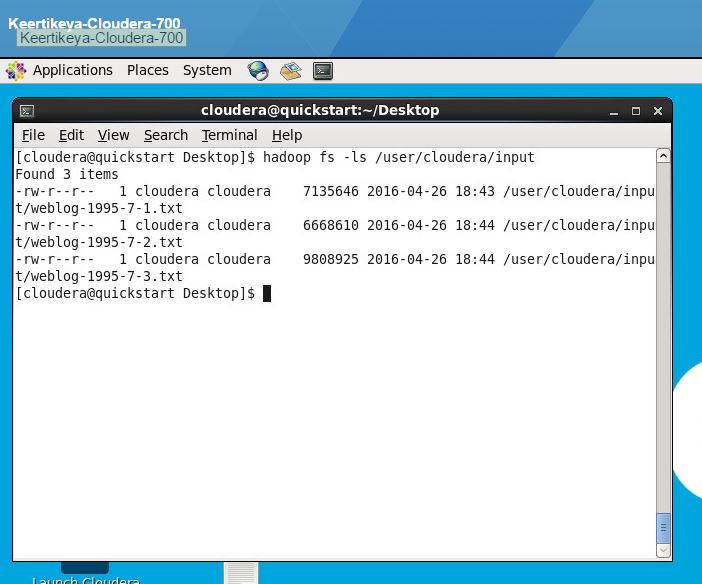
The reducer of this job takes the mapper’s output as its input and aggregates the hit count. This gives us the total hit count of all URIs. The URIs are then written into the output file of the reducer task with URIs and their respective total hit counts.

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.

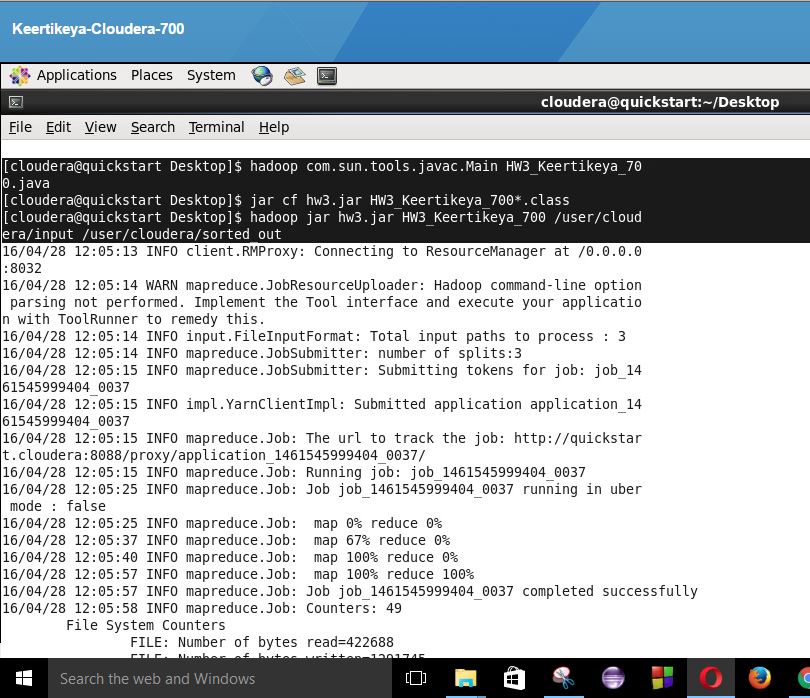
**Q3.**

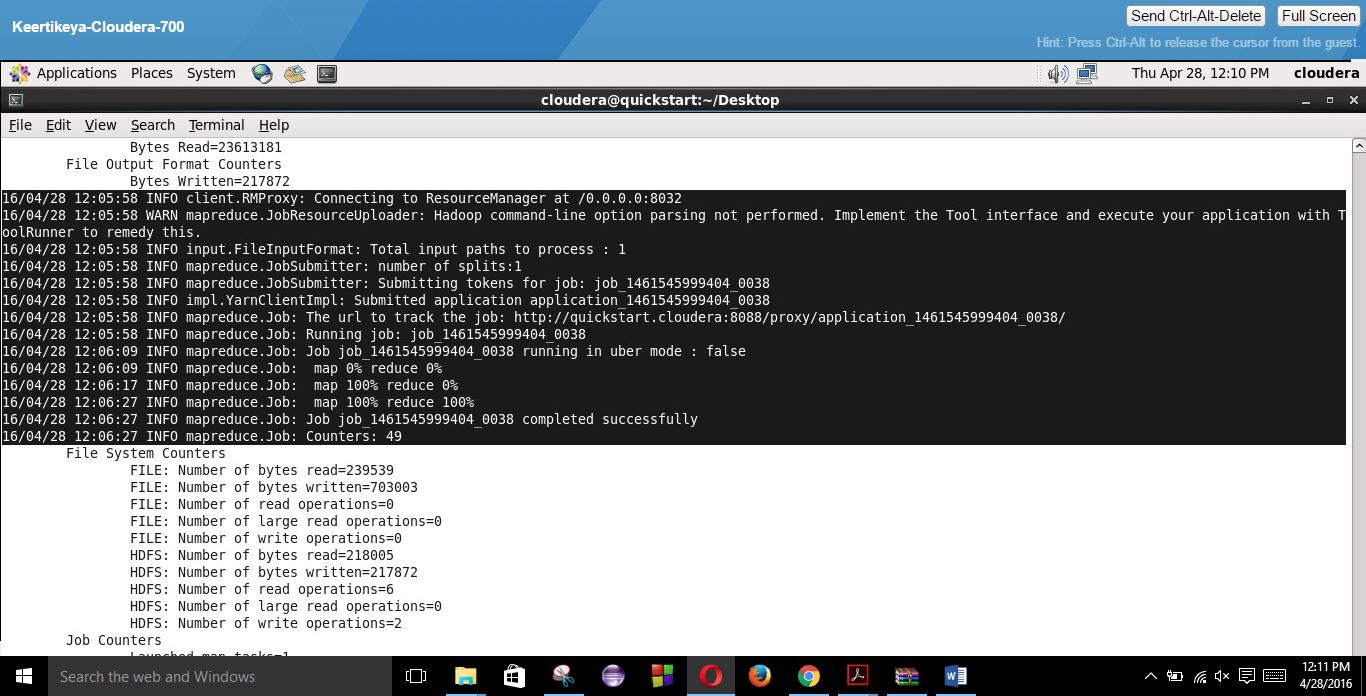
**Ans**

hadoop fs –ls /user/cloudera/input

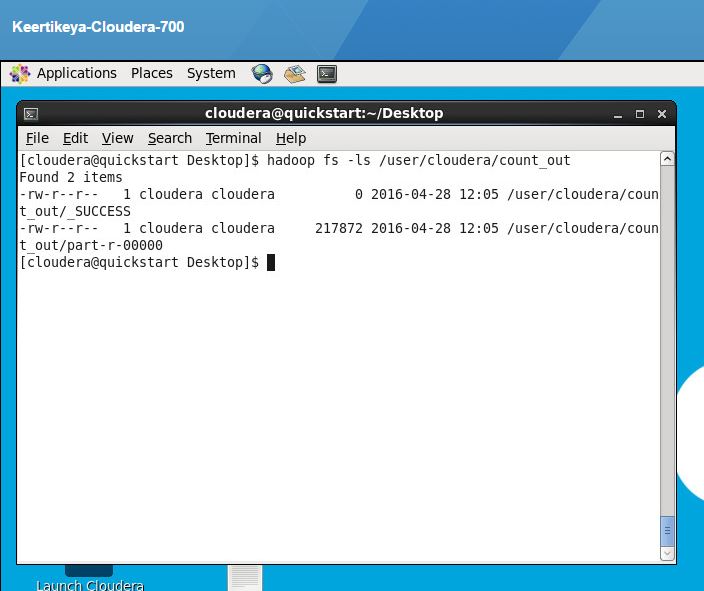


**hadoop jar …**

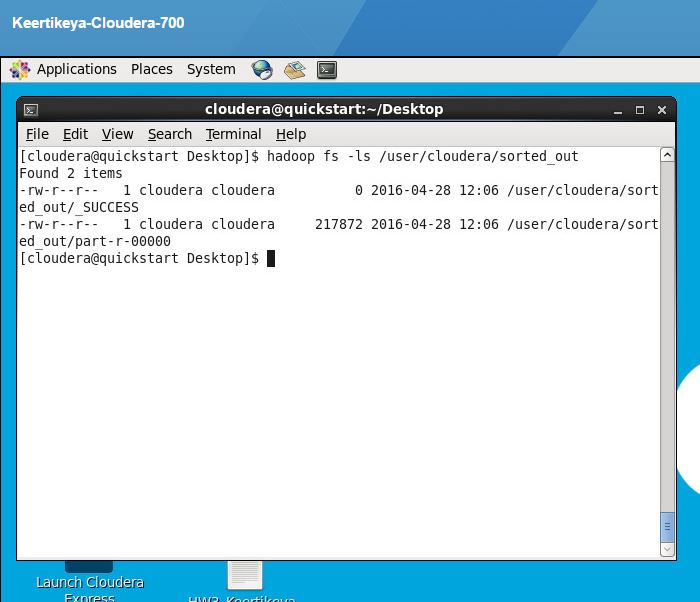




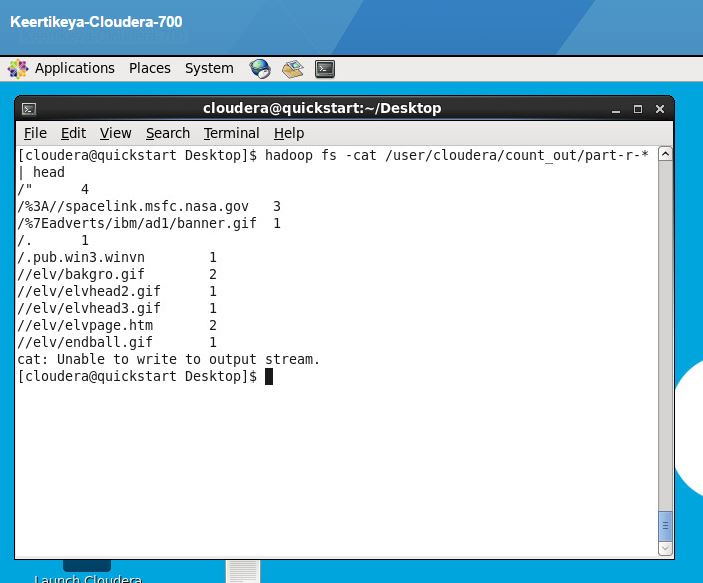
**hadoop fs –ls /user/cloudera/count\_out (output of first MapReduce job)**

****

**hadoop fs –ls /user/cloudera/sorted\_out (final output directory)**

****

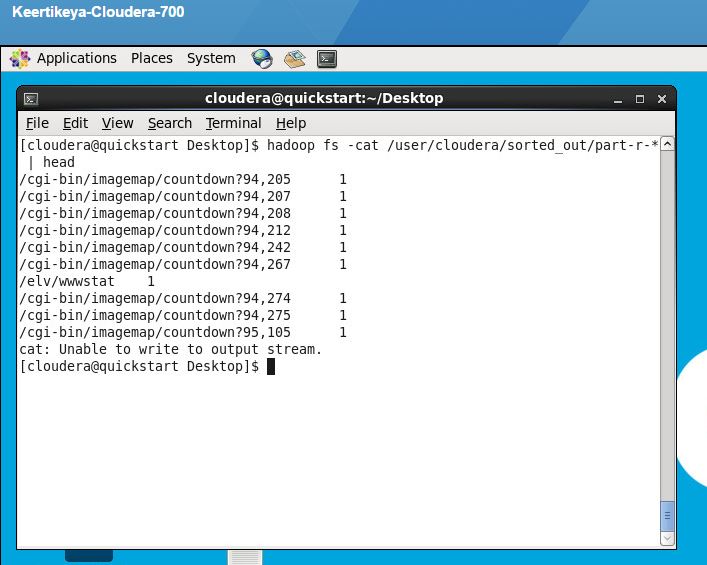
**hadoop fs –cat /user/cloudera/count\_out/part-r-\* | head (first MapReduce job output)**

****

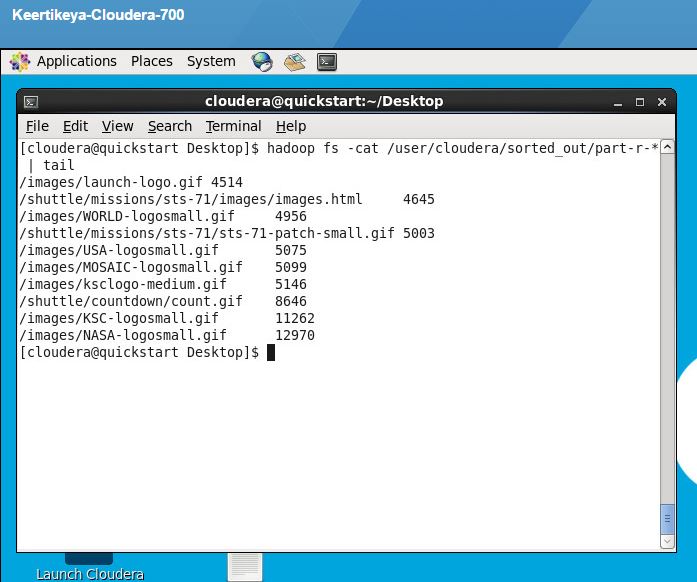
**hadoop fs –cat /user/cloudera/count\_out/part-r-\* | tail (first MapReduce job output)**

****

**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.