Akshay Syal

Write a brief report about your findings, answering the following questions:

1. **[0 points] Did any student in class help you in a major way with the setup process? As a rule of thumb, this would be somebody who spent an hour or more with you, e.g., helping you install Maven and Scala, figuring out why the Makefile did not work for you and so on. Or it could be somebody who saved you at least an hour, e.g., by providing their configuration files. You can credit this student here (at most one person can be credited—if multiple students helped you, only mention the one you think helped you the most), so that this can be taken into account for that person's course-participation credit.**

2. **[28 points] Show the links to your repository with source code etc on Github Classroom.**

a. **Show the link to your MapReduce repository.**
   https://github.com/CS6240/hw1-mr-AkshaySyal

b. **Show the link to your Spark repository.**
   https://github.com/CS6240/hw1-spark-AkshaySyal

3. [20 points] MapReduce Word Count program: Explain in your own words what the map() function is doing. Then explain in your own words what the reduce() function is doing. (a couple of sentences for each)

Ans: In the map() function each word in the text file is mapped to one. This function creates key-value pairs with key as each word in the document and value being 1. That is if the document is:
"Johny Johny Yes Papa" then the output of map function will be:
("Johny",1), ("Johny",1), ("Yes",1), ("Papa",1)

After the map phase all values corresponding to a specific key are grouped together. Reduce function receives the key and list of values having the same key. In this case the key is each word in the input and the list of values is a list of 1s.

In the reduce function the 1s in the list of 1s are added up to count that word's occurrences.

Using the same example as above, output of reduce function will be:
("Johny",2), ("Yes",1), ("Papa",1)

4. [10 points] MapReduce Word Count program: Can you tell from the code in WordCount.java, which line(s) group the output of the map() function by word? If so, point out the specific lines in the source code.

Ans: The grouping of the output of map() function by word is known as Shuffle and Sort phase. The Shuffle and Sort phase is implicitly handled by the MapReduce framework, and you don't see explicit lines of code for Shuffle and Sort in the user-defined classes.

In the program inside the map function the line, “context.write(word, one)” is responsible for emitting the intermediate key-value pairs from the map() function. This key-value pair represents the occurrence of a word, and the MapReduce framework will use these pairs for further processing, including grouping by key during the shuffle and sort phase.

5. [20 points] Spark Word Count program: Which of the Scala commands do the same work as the MapReduce map() function and which do the same work as the reduce() function? What would happen if you used reduce() instead of reduceByKey in the Spark program?

Ans: The flatMap and map scala commands do the same work as MapReduce map() function while the reduceByKey command does the same work as MapReduce reduce() function.

More specifically, the flatMap processes the text file line by line, splits each line into words by spaces and returns a single list containing all the words in the text file.

The map function processes each word in the list returned by flatMap, return or emits (word,1) for each word.

reduceByKey aggregates the counts for each word.

reduce() and reduceByKey() in Spark work differently.

reduce() aggregates the elements in the list using a specified associative function. It just returns a single element. reduceByKey() operates on a key-value pair and performs a reduction operation per key i.e. it groups elements by key and reduces each group separately. It returns a new key-value pair list.

Hence in this program, if reduce() is used instead of reduceByKey then it will throw an error as addition operation can’t be performed on (key,value) pairs.

6. [10 points] For the Spark program, report the lineage of the final result. To obtain it, call counts.toDebugString right before the saveAsTextFile command, and make sure this information is written to your log file, e.g., using logger.info(counts.toDebugString). Copy-and-paste the lines produced by this toDebugString call to answer this question.

Ans:

Relevant log output (standalone mode, local):

```
2024-01-17 17:36:42,740 INFO root: (2) ShuffledRDD[4] at reduceByKey at
WordCount.scala:28 []
 +-(2) MapPartitionsRDD[3] at map at WordCount.scala:27 []
    |  MapPartitionsRDD[2] at flatMap at WordCount.scala:26 []
    |  input MapPartitionsRDD[1] at textFile at WordCount.scala:25 []
    |  input HadoopRDD[0] at textFile at WordCount.scala:25 []
```

7. [5 points] Run your MapReduce and your Spark program on AWS on the input dataset in the project. Use 3 cheap machine instances (1 master and 2 workers). Look for the cheapest machine instance you can select on EMR. In the past these were usually general-purpose machines of type m[?].[*], e.g., m1.small or m5.large. (For some instance types, possibly those from a previous generation, you may need to submit a subnet-id through the Makefile when creating the cluster.) After successful execution, look through the log files (syslog, stderr) it generated to find the following information:

a. **Report the running time of your MapReduce program. To determine it, look for lines in the log file that indicate when the actual MapReduce computation started and finished.**

   Start:
   ```
   2024-01-18 17:37:45,813 INFO org.apache.hadoop.mapreduce.Job (main): Running job:
   job_1705599363973_0001
   ```

   End:
   ```
   2024-01-18 17:38:33,455 INFO org.apache.hadoop.mapreduce.Job (main): Job
   job_1705599363973_0001 completed successfully
   ```

   Running time: 48 seconds

b. **Report the amount of data transferred (in bytes and in records) to the Mappers.**
   Map input records=7368
   Map output bytes=456004

c. **Report the amount of data transferred (in bytes and in records) from Mappers to Reducers.**
   Reduce shuffle bytes=85900
   Map output records=47085

d. **Report the amount of data transferred (in bytes and in records) from Reducers to output.**
   Reduce output records=9566
   Bytes Written=96809

e. **Report the running time of your Spark program.**

   Start: 24/01/18 17:34:34 INFO Client: Application report for application_1705599181935_0001 (state: RUNNING)

   End: 24/01/18 17:34:49 INFO Client: Application report for application_1705599181935_0001 (state: FINISHED)

   Running time: 15 seconds

8. [3 points] MapReduce execution:

a. **Which phase will most likely be the bottleneck in terms of work performed: Map or Reduce? Briefly justify your response.**

   Ans: In this problem the Reduce phase is most likely to be the bottleneck.

   In map phase each mapper processes a portion of the input data independently and produces intermediate key-value pairs (word, count). These map tasks run in parallel and the workload can be evenly distributed.

   In the reduce phase each reducer is processing groups of key-value pairs and aggregating the counts for a particular word. If there are a lot of unique words in the document then a lot of reducers have to be produced. And since aggregation is being performed, this phase involves more computational work than map phase.

b. **How many Map tasks did MapReduce create? Briefly justify your response by exploring the log.**
   By exploring syslog file (AWS run):

```
Job Counters
    ...
    Launched map tasks=1
    ...
```

According to the log, the number of launched map tasks is 1. Therefore, MapReduce created 1 Map task for this job.

c. **How can you determine, how many reduce() function calls your program made? Briefly discuss any ideas.**

```
Job Counters
  ...
  Launched reduce tasks=7
  ...
```

According to the log, the number of launched reduce tasks is 7. Therefore, the program made 7 reduce() function calls.

9. **[4 points] Show the following 4 links:**
   a. **A link to your syslog file created by the run of your MapReduce program for which you reported the running time and data transfer numbers.**
   https://github.com/CS6240/hw1-mr-AkshaySyal/blob/main/syslog.txt

   b. **A link to your output file (or the directory containing the output files if there are multiple) created by the run of your MapReduce program for which you reported the running time and data transfer numbers.**
   https://github.com/CS6240/hw1-mr-AkshaySyal/tree/main/output

   c. **A link to your stderr file created by the run of your Spark program for which you reported the running time.**
   https://github.com/CS6240/hw1-spark-AkshaySyal/blob/main/stderr.txt

   d. **A link to your output file (or the directory containing the output files if there are multiple) created by the run of your Spark program for which you reported the running time.**
   https://github.com/CS6240/hw1-spark-AkshaySyal/tree/main/output