Word Count using AWS EMR

CS6240- Parallel Data Processing
Homework 1

by

Sanchana Mohankumar

A report submitted to a faculty of

Northeastern University

January 2023

Table of Contents:

1. Introduction

MapReduce was developed to manage massive volumes of data, which was called Big Data. A machine, in general, can only manage a certain volume of data at a time due to memory restrictions and hard drive capacity, yet in our society, a large volume of data is generated every day that cannot be handled by a single machine or multiple machines. In today's world map reduce is used in various use cases such Entertainment, E-commerce, social media, Data Warehouse, Fraud Detection etc. Over the years, Map Reduce is still used in practice as it's reliable, Scalable, Secure, fast paced, parallel processing compatible and affordable. Furthermore, we will discuss in elaborate about the process of Map Reduce in detail in section Mapper and Reducer

2. Problem Statement

In this project, we will execute our wordcount code on both local and AWS server.

Task 1: In IntelliJ IDE, we run a wordcount Map Reduce Java program and input our text file to compute the wordcount.

Task 2: We next export the jar file from IntelliJ IDE and save it to Amazon S3 together with the input text file in order to run the EMR cluster that provides the wordcount of the input file.

In this project, we'll examine the differences between a word-counting application running locally and on AWS.

3. Mapper Reducer

At this section, we will examine the workflow and also how Mapper Reducer works in each phase to gain a broad knowledge of the process and how to utilize it.

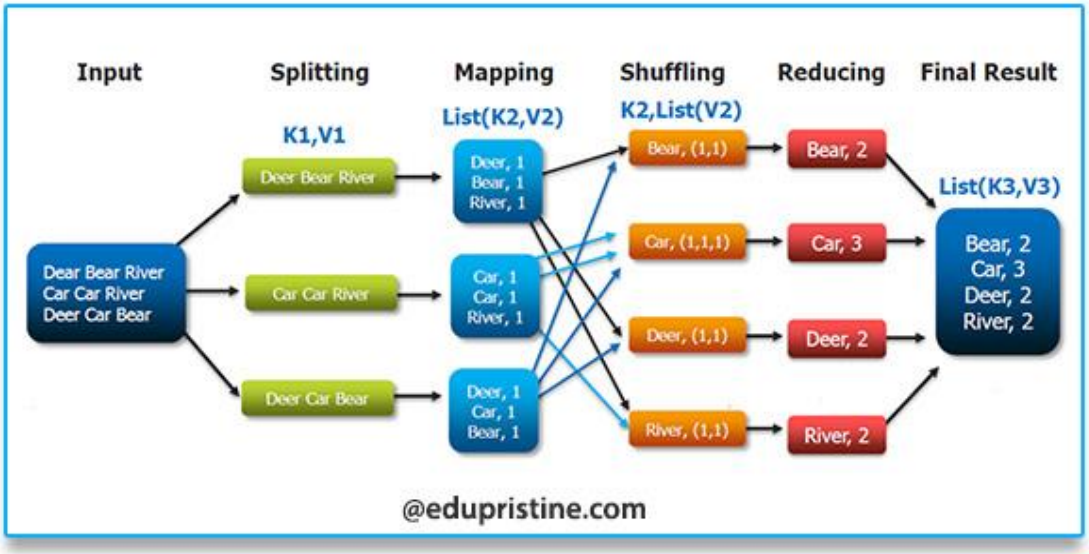


Fig 1: Mapper Reducer process flow

Map Reduce consists of two steps: Map and Reduce. It is a parallel task that is conducted on several distinct computers, one of which is the Map process, which we can design ourselves. Over the process the of Map and Reduce the data gets transformed.

Step 1: We first input our text file

Step 2: The file is split into several independent sentences and assigned to each map process as chunks containing several sentences

Step 3: Each Map process splits into key value pair where the key is words and value are 1, in the above figure 1 K is key and V is value

Step 4: Before moving to the Reducing phase, the Map process assigns a value of one to each key, which in our case is word. Following that, shuffling occurs, in which all identical words are grouped together, resulting in distinct key words and storing all the value as a list for each key. In figure 1 above, during the shuffling process, each key displayed is distinct, and the values are stored as a list.

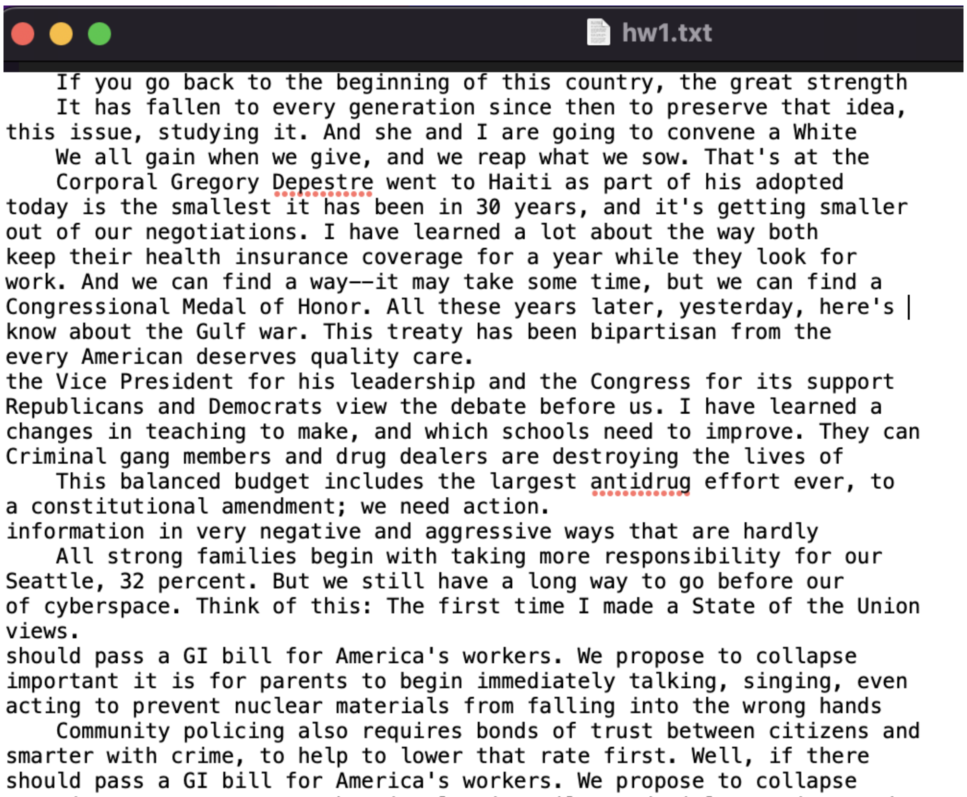
Step 5: During the Reducing process, all the values for each key, which in our case is word, are summed together. As we can see in the above figure 1, key Bear has a value of 2, and so do car, deer, and River.

Step 6: Finally, all the data from the Reducer process is typically extracted in a single file as a distinct key with its value, resulting in data transformation.

The above steps showcase an overall idea about the Map Reduce process, further lets manually execute the above steps and check out the results.

4. Data

In this Project we are used a text file as input, Figure 2 below is the text file which we used for this project. The text file contains a data size of 1.45GB.


hw1.txt

```
    If you go back to the beginning of this country, the great strength
    It has fallen to every generation since then to preserve that idea,
this issue, studying it. And she and I are going to convene a White
    We all gain when we give, and we reap what we sow. That's at the
    Corporal Gregory Depestre went to Haiti as part of his adopted
today is the smallest it has been in 30 years, and it's getting smaller
out of our negotiations. I have learned a lot about the way both
keep their health insurance coverage for a year while they look for
work. And we can find a way--it may take some time, but we can find a
Congressional Medal of Honor. All these years later, yesterday, here's
know about the Gulf war. This treaty has been bipartisan from the
every American deserves quality care.
the Vice President for his leadership and the Congress for its support
Republicans and Democrats view the debate before us. I have learned a
changes in teaching to make, and which schools need to improve. They can
Criminal gang members and drug dealers are destroying the lives of
    This balanced budget includes the largest antidrug effort ever, to
a constitutional amendment; we need action.
information in very negative and aggressive ways that are hardly
    All strong families begin with taking more responsibility for our
Seattle, 32 percent. But we still have a long way to go before our
of cyberspace. Think of this: The first time I made a State of the Union
views.
should pass a GI bill for America's workers. We propose to collapse
important it is for parents to begin immediately talking, singing, even
acting to prevent nuclear materials from falling into the wrong hands
    Community policing also requires bonds of trust between citizens and
smarter with crime, to help to lower that rate first. Well, if there
should pass a GI bill for America's workers. We propose to collapse
```

Fig 2: Text file used for the project

5. Local Execution

Task 1: In IntelliJ IDE, we ran a wordcount Map Reduce Java program and input our text file to compute the wordcount.

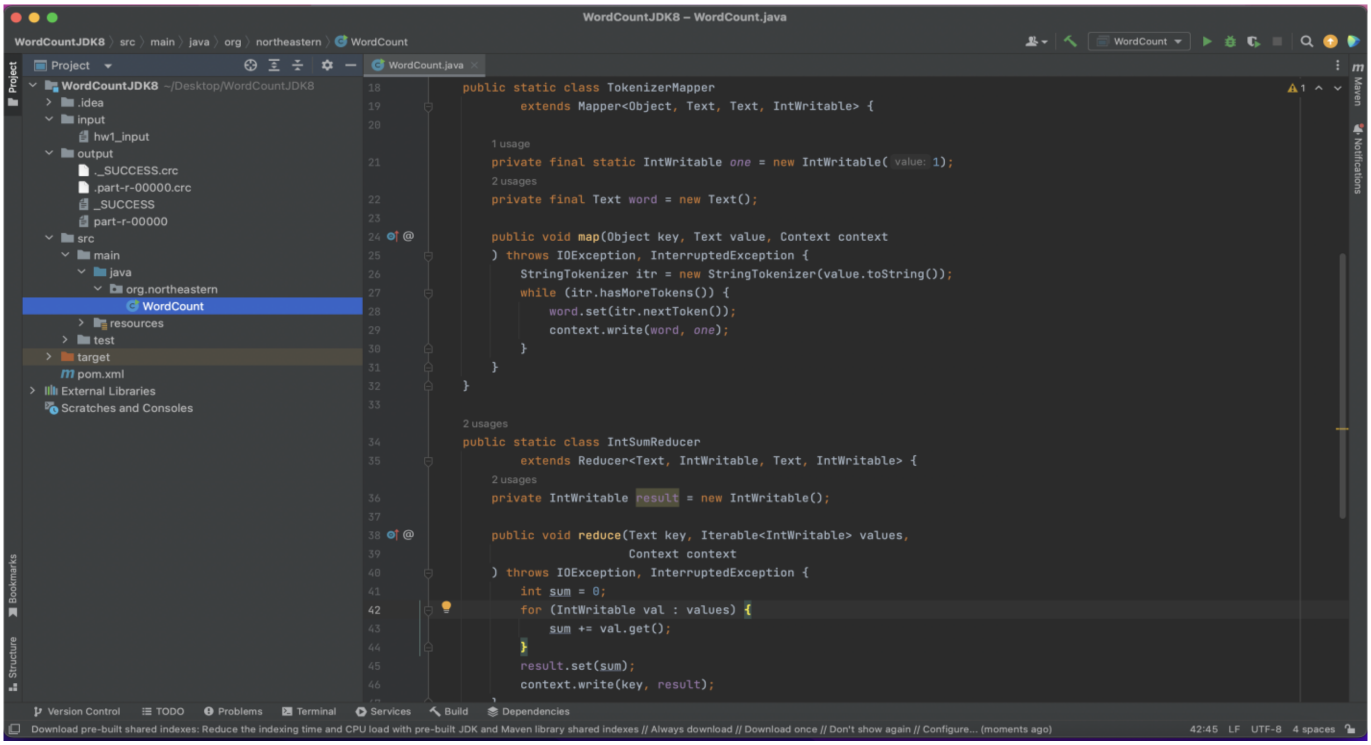


Image 1: Project directory structure showing Wordcount Java file

In the above Image 1, we can see the Project directory structure of the Word Count Java file.

Step 1: Created a project named WordcountJDK8 with Language as Java version 8 and build system as Maven

Step 2: Updated pom.xml with dependencies of Maven, Hadoop and Java

Step 3: Under Project WordcountJDK8 select src → main → java -> package and named it as org.northeastern

Step 4: After creating the package we add a Java class and add the wordcount Map Reduce Java program

Step 5: Under project we add another folder input and add our text file

Step 6: Once the input file got added under edit configuration, I changed the argument to input output which generates an output folder with wordcount output after running wordcount java file

Step 7: As we can see in Image 2, showing output files under the folder output of which we have displayed the part-r-00000 which shows the words and its number of occurrences

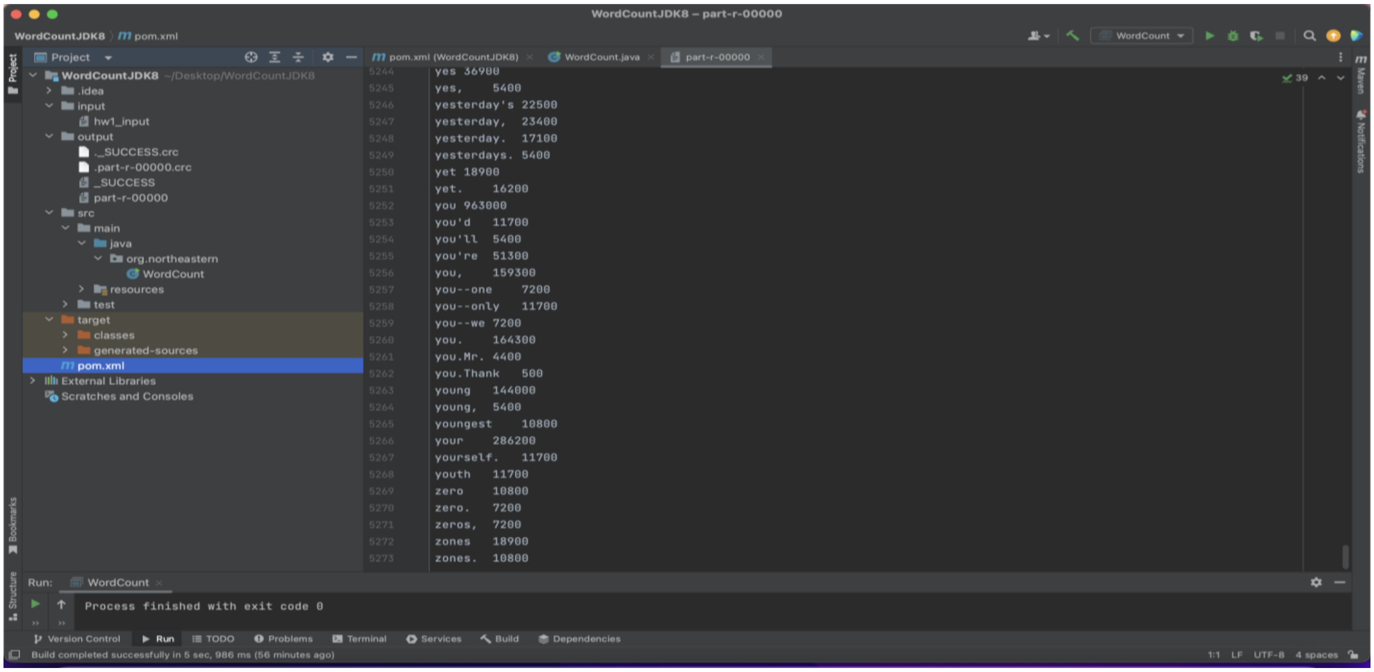


Image 2: Project directory structure showing output file displaying last 30-word count

In Image 1, we can see a Word Count Java file having two class TokenizerMapper, IntSumReducer which carries function of Mapper and Reducer

- TokenizerMapper has four inputs to the class, which are input text files that are key values which are object and Text, text for output, and Intwritable is the value. Again, a map function is created with input as key object, Text value, and context, which aids in the separation of the key value pair. Furthermore, the text is tokenized using Java string Tokenizer, and the attribute Intwrittable is assigned a value of 1. Finally, in the loop, the entire text file is tokenized, and a key value pair is created.
- IntSumReducer has 4 inputs to the class Text and Intwrittable which are the key and value input from Mapper and another set of Text and Intwrittable which is for output of key and summed value. Again, a reduce function is created with input as text and inwrittable for list of values and context for writing out output. Finally, the values are iterated and summed up with respect to key values

In image 3, we are executing the below output using console

The below image says no of input files to process is 1 and is split into 11 parts and after the mapper is processed is 100% the reducer completes the process.

- File System counter and Job Counters counts the number of bytes read by the file system and the number of bytes written which helps in analyzing MapReduce job quality and for application level.
- Map Reduce Framework explains the statistics of mapper process and shuffling process as explained in Mapper and Reducer section
- File input format and output format we can see the no of bytes read and decrease in output bytes as we extracted the distinct words and calculated the count

```
2023-01-28 13:09:14,799 INFO input.FileInputFormat: Total input files to process : 1
2023-01-28 13:09:15,801 INFO mapreduce.JobSubmitter: number of splits:11
2023-01-28 13:09:16,361 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1674373701935_0006
2023-01-28 13:09:16,362 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-01-28 13:09:16,430 INFO conf.Configuration: resource-types.xml not found
2023-01-28 13:09:16,430 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-01-28 13:09:16,770 INFO impl.YarnClientImpl: Submitted application application_1674373701935_0006
2023-01-28 13:09:16,784 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1674373701935_0006/
2023-01-28 13:09:16,784 INFO mapreduce.Job: Running job: job_1674373701935_0006
2023-01-28 13:09:22,906 INFO mapreduce.Job: Job job_1674373701935_0006 running in uber mode : false
2023-01-28 13:09:22,910 INFO mapreduce.Job:  map 0% reduce 0%
2023-01-28 13:09:39,298 INFO mapreduce.Job:  map 31% reduce 0%
2023-01-28 13:09:42,346 INFO mapreduce.Job:  map 55% reduce 0%
2023-01-28 13:09:56,610 INFO mapreduce.Job:  map 64% reduce 0%
2023-01-28 13:09:57,629 INFO mapreduce.Job:  map 86% reduce 0%
2023-01-28 13:09:58,658 INFO mapreduce.Job:  map 100% reduce 0%
2023-01-28 13:09:59,671 INFO mapreduce.Job:  map 100% reduce 100%
2023-01-28 13:10:01,698 INFO mapreduce.Job: Job job_1674373701935_0006 completed successfully
2023-01-28 13:10:01,745 INFO mapreduce.Job: Counters: 51
        File System Counters
                FILE: Number of bytes read=7190934
                FILE: Number of bytes written=11314449
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1454049978
                HDFS: Number of bytes written=72815
                HDFS: Number of read operations=38
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=12
                Launched reduce tasks=1
                Data-local map tasks=12
                Total time spent by all maps in occupied slots (ms)=187171
                Total time spent by all reduces in occupied slots (ms)=14726
```

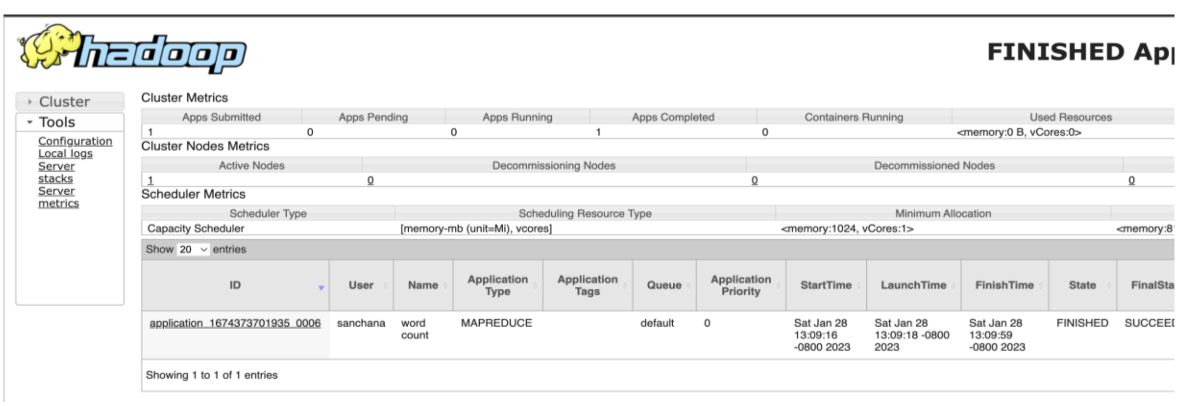
```
        Total time spent by all map tasks (ms)=187171
        Total time spent by all reduce tasks (ms)=14726
        Total vcore-milliseconds taken by all map tasks=187171
        Total vcore-milliseconds taken by all reduce tasks=14726
        Total megabyte-milliseconds taken by all map tasks=191663104
        Total megabyte-milliseconds taken by all reduce tasks=15079424
    Map-Reduce Framework
        Map input records=21907700
        Map output records=248943500
        Map output bytes=2418234700
        Map output materialized bytes=816409
        Input split bytes=1518
        Combine input records=249396411
        Combine output records=510914
        Reduce input groups=5273
        Reduce shuffle bytes=816409
        Reduce input records=58003
        Reduce output records=5273
        Spilled Records=568917
        Shuffled Maps =11
        Failed Shuffles=0
        Merged Map outputs=11
        GC time elapsed (ms)=3502
        CPU time spent (ms)=0
        Physical memory (bytes) snapshot=0
        Virtual memory (bytes) snapshot=0
        Total committed heap usage (bytes)=3620732928
    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=1454048460
    File Output Format Counters
        Bytes Written=72815
```

Image 3: Console Output

After the above step is completed a local host URL is created to check the status of the Job as we can see in the below image 4

User = **sanchana ,** Application Type = **MAPREDUCE,** state of the job = **FINISHED** and Final Status = **SUCCEEDED**

So, we can confirm from the below local host URL generated from running the program that the task is successful



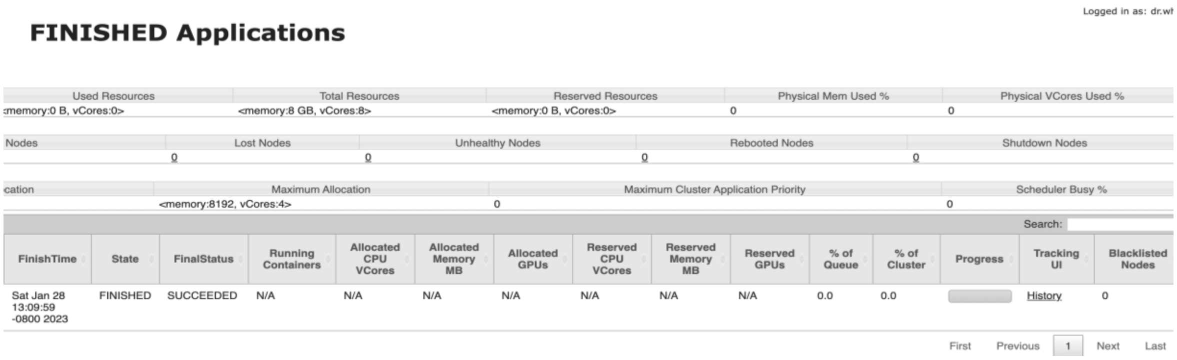


Image 4: Successful execution of wordcount using Hadoop

6. AWS Execution

Task 2: We next export the jar file from IntelliJ IDE and save it to Amazon S3 together with the input text file in order to run the EMR cluster that provides the wordcount of the input file.

JAR File

Steps to extract Jar file

Step 1: File -> Project Structure -> Project Seting -> Artifacts -> + Add -> Jar -> From modules with dependencies -> Main Class –> wordcount -> Extract to target Jar -> Press ok

Step 2: Build -> Build Artifact -> Action-> Build

AWS EMR

Step 1: Created an AWS account

Step 2: Created a security key pair to control communication and information passed around cluster. On EC2 Dashboard under Network and Security we select key pairs and created a new key pair

Step 3: Created an Amazon S3 bucket to store the input text file and jar file taken from the IntelliJ IDE.

Step 4: Navigating to EMR Dashboard we create a cluster and configure the cluster by selecting Hadoop Framework, add the key pair we created for security and assign the primary and core nodes necessary for the process

Step 5: Once the cluster was created, I added a step under steps and assigned jar file and input file path, and output location path for the output file to get stored

Step 6: After the Status is complete. The output file is created in Amazon S3 under the bucket displaying the output word and its values

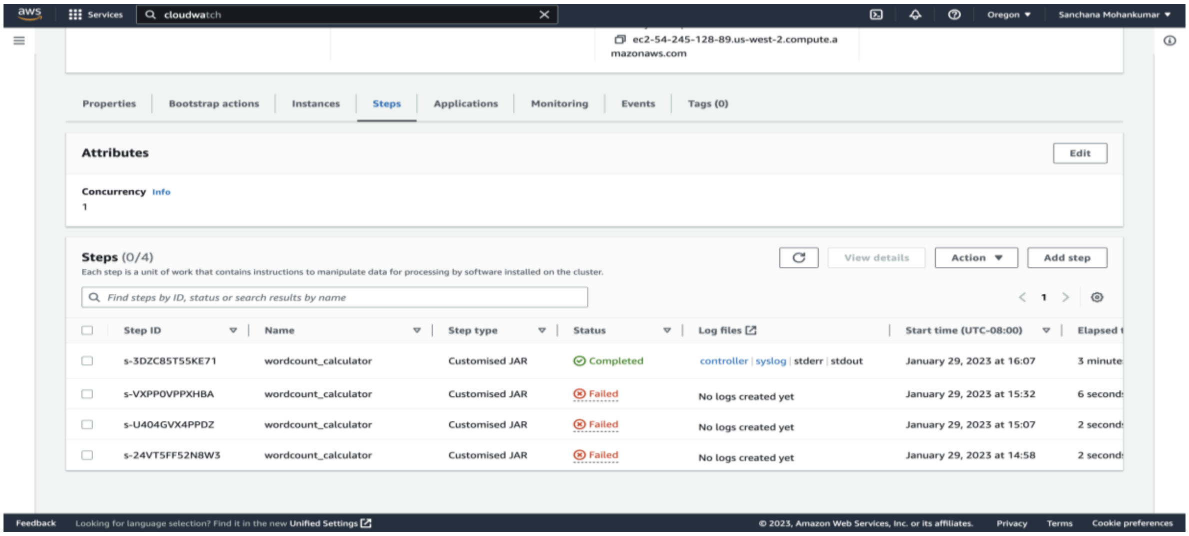


Image 5: AWS EMR status using Customized Jar

Image 5 displays output of Step 5 where the status shows completed signifies successful completion of word count task using Jar file and input text file extracted from Amazon S3.

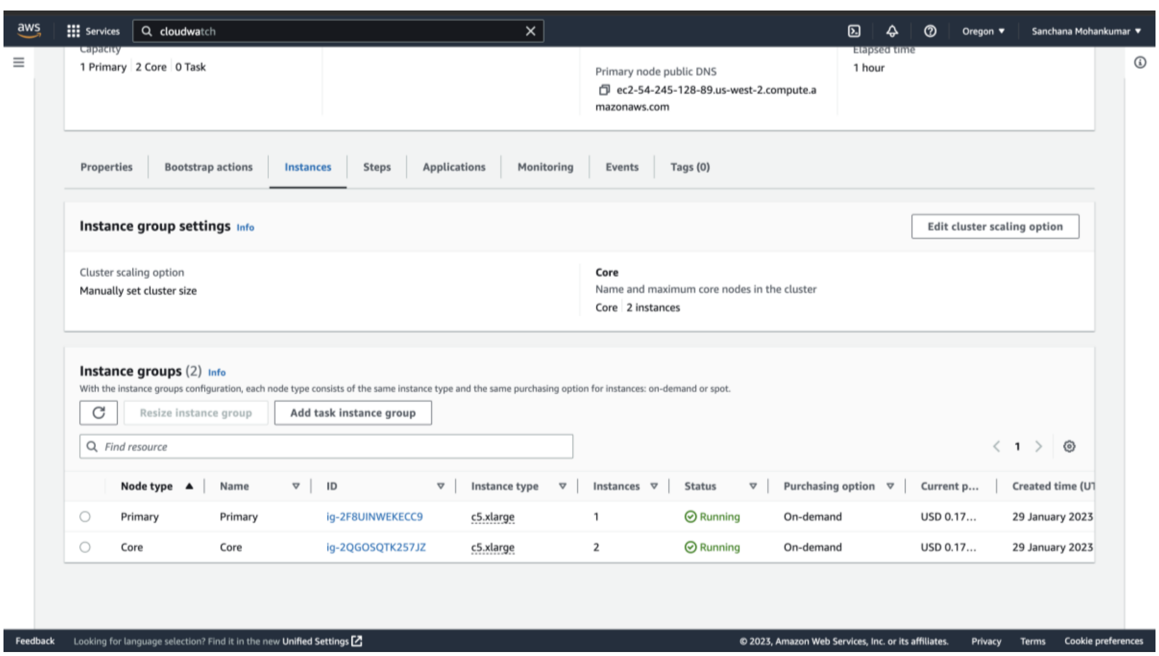


Image 6: AWS EMR – Status of 1 Primary Node and 2 Core Node

Image 6 shows the successful status of both Primary and Core Nodes. Under instances we can see 1 Primary node and 2 core nodes that we assigned during formation of cluster

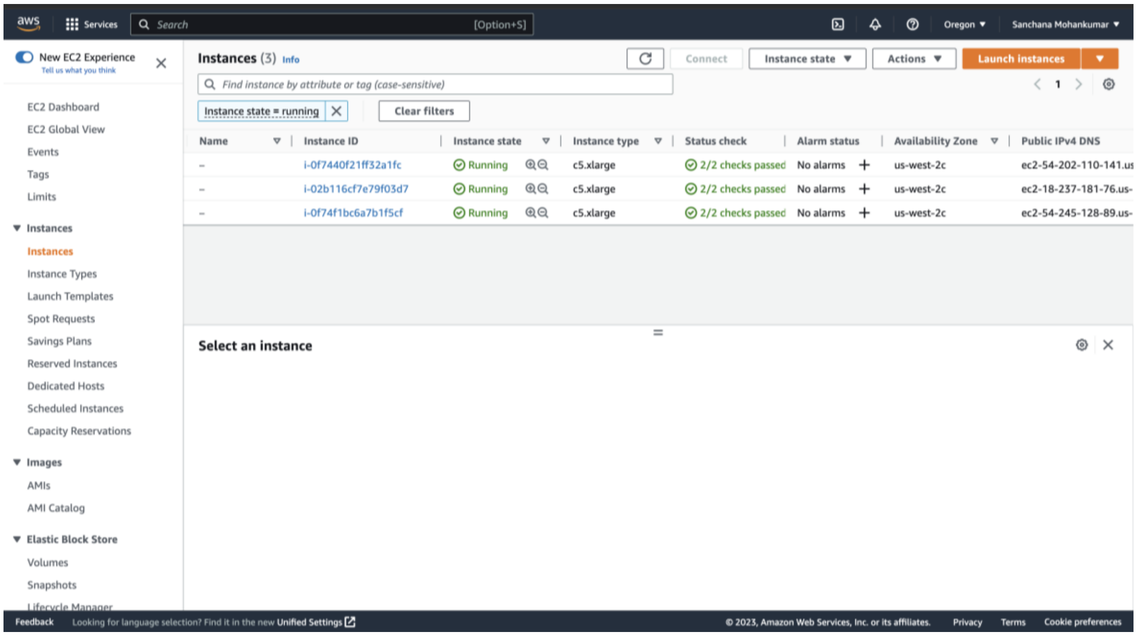


Image 7: Primary and Core nodes status

Image 7 shows the successful instance status of all the primary and Core Nodes

7. Conclusion

As our computing demands vary, Amazon EMR gives us the ability to scale the cluster up or down. We may also change the size of our cluster to add more instances for peak workloads and remove more instances when peak workloads decrease to cut expenses. If a system malfunctions, it is automatically replaced, and precise log files are also maintained. However, with local implementation, we are not given the privilege of quickly adapting solutions to meet our requirements.

8. GitHub Link

https://github.com/Sanchana1997/WordCount-using-AWS-

I have attached the GitHub link for the project. Also, have included the output files and log files generated from AWS EMR.

9. Reference:

https://github.com/autopear/Intellij-Hadoop

https://towardsdatascience.com/installing-hadoop-on-a-mac-ec01c67b003c

https://www.youtube.com/watch?v=JDk-LYJMzEU

https://www.spiceworks.com/tech/big-data/articles/what-is-map-reduce/