Module 5 Assignment – Individual Lab

Professor: Daya Rudhramoorthi

Course: ALY6110: Data Management and Big Data (CRN 70590)

Submitted by :

Sameeksha Bellavara Santhosh

Date: 22-10-2023

INTRODUCTION

Apache Spark is an open-source distributed computing framework created to be fast, user-friendly, and adept at sophisticated data analyses. It facilitates the programming of entire clusters with built-in data parallelism and fault-tolerance. Developed to streamline big data processing and enhance its speed, Spark has carved a niche for itself in the big data arena. It's a comprehensive platform that caters to diverse needs like batch processing, real-time processing, machine learning, and graph analytics.

This task went beyond a mere word count challenge. It encapsulated various nuances of Apache Spark and big data analytics:

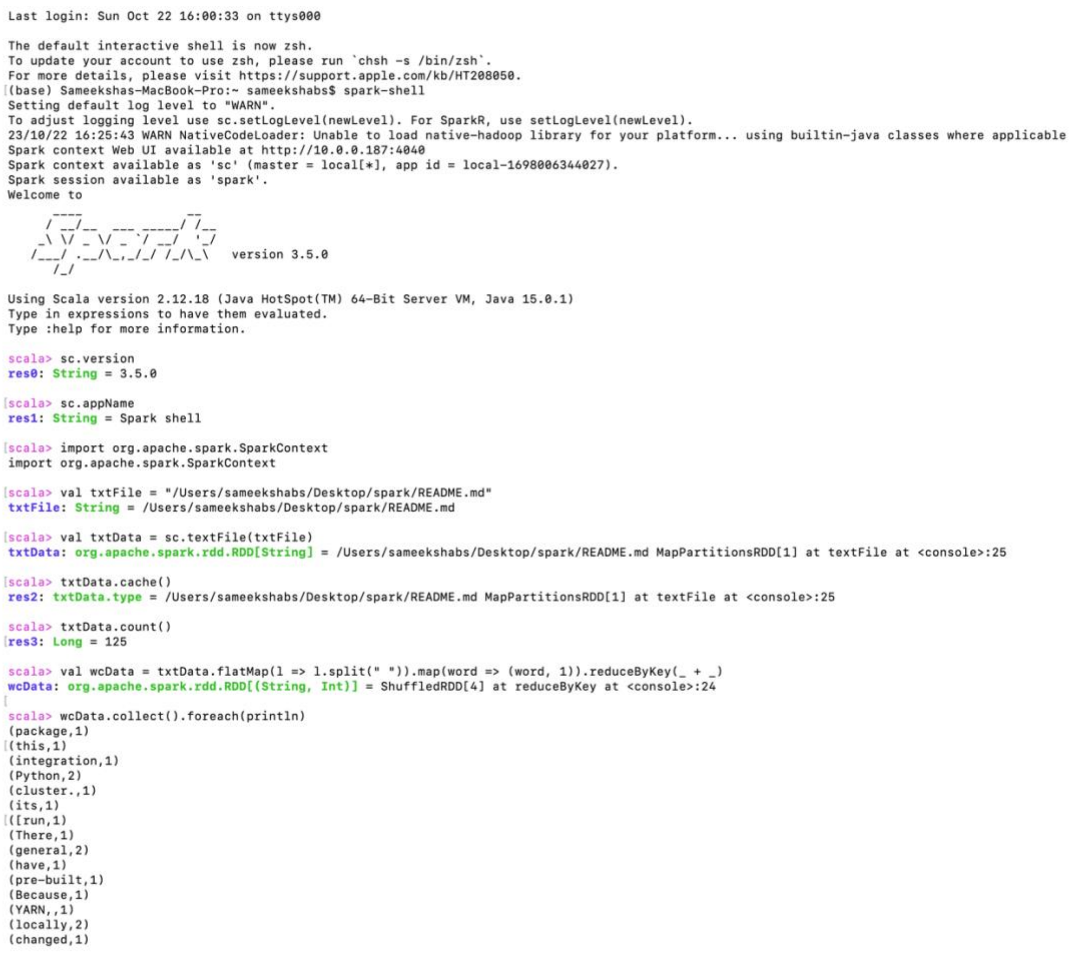
Data Processing Skills :
• By assessing a textual document for word occurrence frequencies, the exercise put on display Spark's prowess in dissecting and managing data, a cornerstone of big data analytics.
Spark Scala Shell Interactivity
• Using the Spark Scala Shell underscored Spark's capacity for on-the-fly data probing. This allows for immediate testing, debugging, and modifications of data tasks without a full application setup.
Embrace of MapReduce :
• At its core, the exercise leveraged the MapReduce model, illustrating the process of translating data into key-value configurations and then consolidating them to gain knowledge. This methodology is fundamental to expansive data handling.
Performance Monitoring via Spark Web UI :
• A snapshot from the Spark Web UI shed light on the performance dynamics of the Spark operations. This highlights the significance of keeping an eye on and refining Spark tasks for practical applications.
Real-world Relevance :
• Though elementary, the exercise mirrors practical applications where entities scrutinize extensive textual datasets for insights, from discerning public sentiment and spotting trends to condensing content.

The Spark Scala Shell offers an interactive platform to execute Spark commands in Scala directly. Essentially, it's an immediate-response environment tailored for Spark, streamlining the process for developers to assess and rectify their Spark programs.

For this task, the Spark Scala Shell was employed to conduct text analysis on the README.md document.

In the Spark Scala Shell, the procedure was executed in the following manner:

• **File Input:** The README.md document was ingested into an RDD, which stands for Resilient Distributed Dataset. RDDs are the backbone of Spark, enabling data distribution throughout a cluster for simultaneous processing.



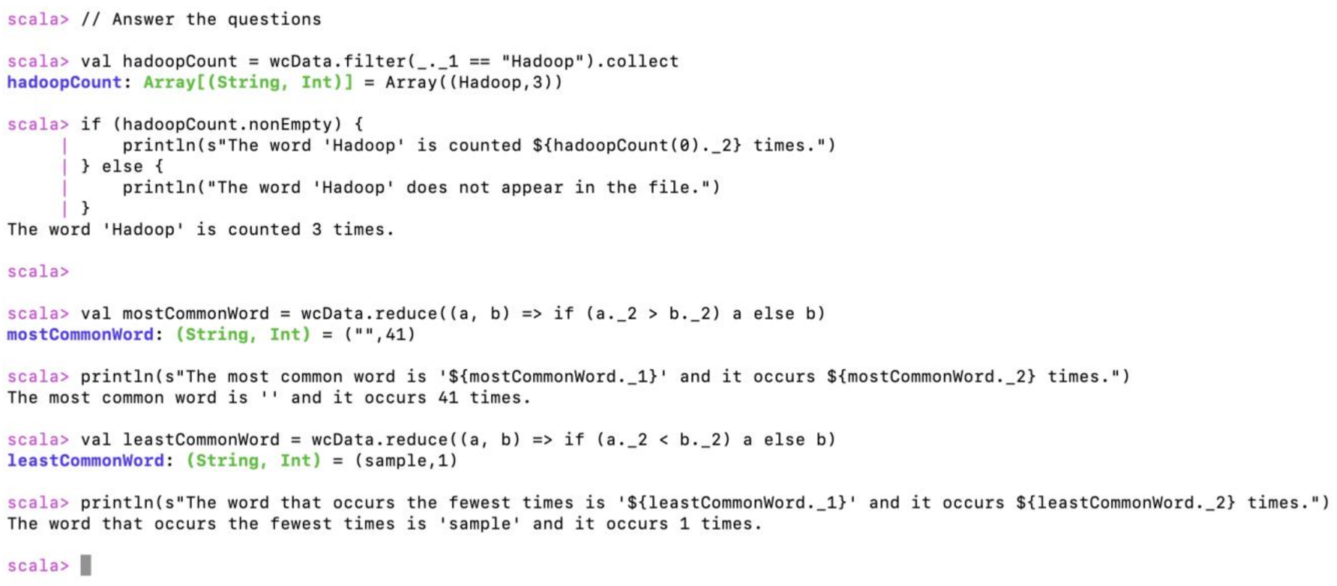
```
Last login: Sun Oct 22 16:00:33 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Sameekshas-MacBook-Pro:~ sameekshabs$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/22 16:25:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.0.0.187:4040
Spark context available as 'sc' (master = local[*], app id = local-1698006344027).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 15.0.1)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc.version
res0: String = 3.5.0

scala> sc.appName
res1: String = Spark shell

scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> val txtFile = "/Users/sameekshabs/Desktop/spark/README.md"
txtFile: String = /Users/sameekshabs/Desktop/spark/README.md

scala> val txtData = sc.textFile(txtFile)
txtData: org.apache.spark.rdd.RDD[String] = /Users/sameekshabs/Desktop/spark/README.md MapPartitionsRDD[1] at textFile at <console>:25

scala> txtData.cache()
res2: txtData.type = /Users/sameekshabs/Desktop/spark/README.md MapPartitionsRDD[1] at textFile at <console>:25

scala> txtData.count()
res3: Long = 125

scala> val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wcData: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:24

scala> wcData.collect().foreach(println)
(package,1)
(this,1)
(integration,1)
(Python,2)
(cluster.,1)
(its,1)
([run,1)
(There,1)
(general,2)
(have,1)
(pre-built,1)
(Because,1)
(YARN,,1)
(locally,2)
(changed,1)
```

• **Breaking Down and Refining**: The document's content was segmented into individual words. All characters that weren't alphanumeric were discarded, and every word was standardized to lowercase to maintain uniformity during the tallying process.

• **Counting Words:** The mapReduce method was employed here. Each distinct word was associated with a key-value combination (with the specific word being the key and the numeral 1 as its value). These combinations were subsequently grouped by their keys, consolidating the counts for every distinct word. The count of the text data is 125 as mentioned in the above snapshot.

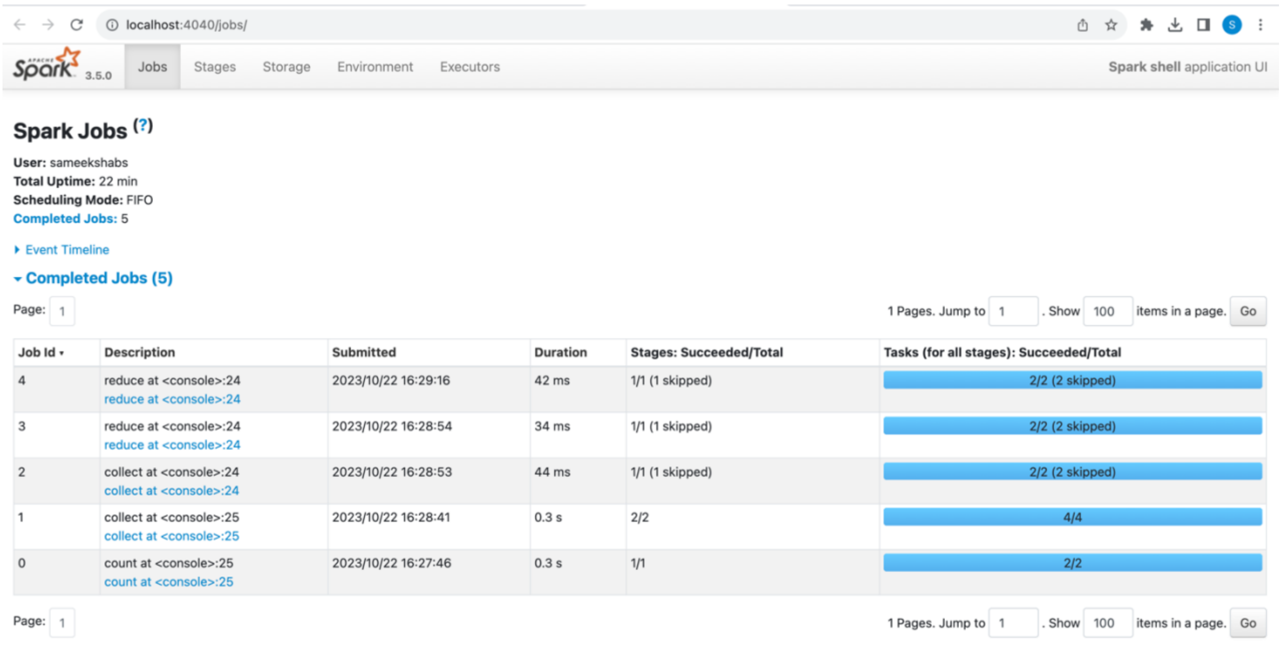
Answering the set of Questions:

Answer 1: The term "Hadoop" is mentioned 3 times in the README.md document.
Answer 2: The word that appears most frequently seems to be an empty string that is " ", possibly representing spaces or non-letter/number symbols, with 41 instances. This could be a result of how the data was processed or presented in the shell.
Answer 3: The word that occurs the fewest times is 'sample', and it occurs only once.



```
scala> // Answer the questions

scala> val hadoopCount = wcData.filter(_._1 == "Hadoop").collect
hadoopCount: Array[(String, Int)] = Array((Hadoop,3))

scala> if (hadoopCount.nonEmpty) {
     |     println(s"The word 'Hadoop' is counted ${hadoopCount(0)._2} times.")
     | } else {
     |     println("The word 'Hadoop' does not appear in the file.")
     | }
The word 'Hadoop' is counted 3 times.

scala>

scala> val mostCommonWord = wcData.reduce((a, b) => if (a._2 > b._2) a else b)
mostCommonWord: (String, Int) = ("",41)

scala> println(s"The most common word is '${mostCommonWord._1}' and it occurs ${mostCommonWord._2} times.")
The most common word is '' and it occurs 41 times.

scala> val leastCommonWord = wcData.reduce((a, b) => if (a._2 < b._2) a else b)
leastCommonWord: (String, Int) = (sample,1)

scala> println(s"The word that occurs the fewest times is '${leastCommonWord._1}' and it occurs ${leastCommonWord._2} times.")
The word that occurs the fewest times is 'sample' and it occurs 1 times.

scala> ▌
```

Spark web Console:
The Spark Web UI provides a glimpse into the operational details and efficiency indicators of Spark programs. In the context of this task, the Web UI displayed the time span of the Spark session.



Spark Jobs (?)

User: sameekshabs
Total Uptime: 22 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	reduce at <console>:24 reduce at <console>:24	2023/10/22 16:29:16	42 ms	1/1 (1 skipped)	2/2 (2 skipped)
3	reduce at <console>:24 reduce at <console>:24	2023/10/22 16:28:54	34 ms	1/1 (1 skipped)	2/2 (2 skipped)
2	collect at <console>:24 collect at <console>:24	2023/10/22 16:28:53	44 ms	1/1 (1 skipped)	2/2 (2 skipped)
1	collect at <console>:25 collect at <console>:25	2023/10/22 16:28:41	0.3 s	2/2	4/4
0	count at <console>:25 count at <console>:25	2023/10/22 16:27:46	0.3 s	1/1	2/2

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Examining the Spark Web UI is crucial for enhancing performance. The interface offers a detailed view of every phase in the application, showcasing tasks, their timeframes, and possible performance hindrances. From the given snapshot, the Web UI showed the duration of 22 min the Spark session required to finalize the word count operations on the README.md document.

REFERENCES

1. Penchikala, S. (2015, January 30). Big Data Processing with Apache Spark – Part 1: Introduction. *InfoQ*. https://www.infoq.com/articles/apache-spark-introduction%20/
2. Tan, P. S. (2022, March 30). Running a MapReduce Word Count application in Docker using Python SDK. *Medium*. https://betterprogramming.pub/running-a-mapreduce-word-count-application-in-docker-using-python-sdk-ed2f1e27d424