Assignment 4: Implement Data Processing using Spark and Compare with MapReduce

Objective: Develop in-memory analytics capability by implementing the same data processing task using Apache Spark and comparing performance with MapReduce.

4.1 Step 1: Start Hadoop Services

```
su - hadoop
start-dfs.sh
start-yarn.sh
jps
```

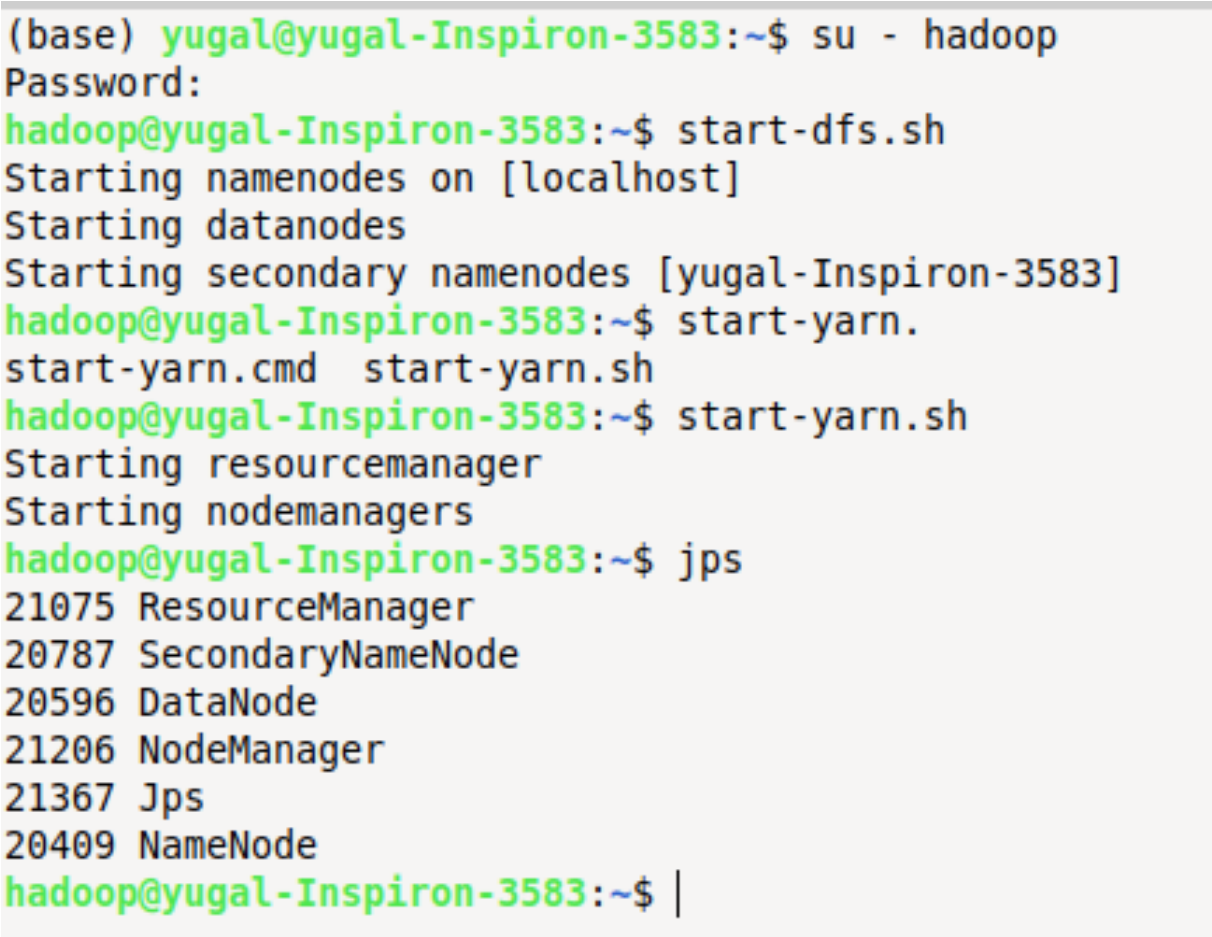

Figure 43: Starting Hadoop services and verifying using jps

4.2 Step 2: Verify Input File in HDFS

The same dataset used in MapReduce is reused for Spark processing.

```
hdfs dfs -ls /
hdfs dfs -ls /input
hdfs dfs -cat /input/input.txt
```

```
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /
Found 4 items
drwxr-xr-x   - hadoop supergroup          0 2026-01-28 01:01 /data
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /input
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /output
drwx------   - hadoop supergroup          0 2026-02-10 14:44 /tmp
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /input
Found 1 items
-rw-r--r--   1 hadoop supergroup         89 2026-02-10 14:52 /input/input.txt
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -cat /input/input.txt
hadoop mapreduce hadoop
big data hadoop mapreduce
mapreduce big data
hadoop big big data
hadoop@yugal-Inspiron-3583:~$
```

Figure 44: Verifying input dataset in HDFS

4.3 Step 3: Start Spark Shell

```
~/spark-3.5.1-bin-hadoop3/bin/spark-shell
```

Spark shell starts successfully and provides the Scala prompt for interactive execution.

```
hadoop@yugal-Inspiron-3583:~$ ~/spark-3.5.1-bin-hadoop3/bin/spark-shell
26/02/18 18:15:50 WARN Utils: Your hostname, yugal-Inspiron-3583 resolves to a loopback address: 127.0.1.1; using 10.42.0.1 instead (on interface wlp2s0)
26/02/18 18:15:50 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/18 18:16:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.42.0.1:4040
Spark context available as 'sc' (master = local[*], app id = local-1771418770571).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 11.0.30)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 45: Launching Spark Shell

4.4 Step 4: Load Data from HDFS into Spark

```
val lines = sc.textFile("hdfs://localhost:9000/input/input.txt")
lines.collect()
```

The dataset stored in HDFS is loaded into an RDD for processing.

```
scala> val lines = sc.textFile("hdfs://localhost:9000/input/input.txt")
lines: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/input/input.txt MapPartitionsRDD[1] at textFile at <console>:23

scala>
```

Figure 46: Reading input data from HDFS

4.5 Step 5: Implement Spark Transformations

Word Count is implemented using Spark transformations.

```
val words = lines.flatMap(line => line.split(" "))
val wordPairs = words.map(word => (word, 1))
val counts = wordPairs.reduceByKey((a,b) => a+b)
```

flatMap() splits lines into words, **map()** creates key-value pairs, and **reduceByKey()** aggregates counts.

```
scala> val words = lines.flatMap(line => line.split(" "))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> val wordPairs = words.map(word => (word, 1))
wordPairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:23

scala> val counts = wordPairs.reduceByKey((a,b) => a+b)
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:23
```

Figure 47: Spark transformations for Word Count

4.6 Step 6: Apply Actions and Display Output

```
counts.collect()
```

Output obtained:

```
(big,4)
(data,3)
(mapreduce,3)
(hadoop,4)
```

```
scala> counts.collect()
res2: Array[(String, Int)] = Array((big,4), (data,3), (mapreduce,3), (hadoop,4))
```

Figure 48: Displaying results using collect()

4.7 Step 7: Save Output to HDFS

```
counts.saveAsTextFile("hdfs://localhost:9000/output_spark")
```

The processed output is stored in HDFS similar to MapReduce output.

```
scala> counts.saveAsTextFile("hdfs://localhost:9000/output_spark")

scala> :quit
```

Figure 49: Saving Spark output to HDFS

4.8 Step 8: Check Output Directory and Validate Result

```
hdfs dfs -ls /
hdfs dfs -cat /output_spark/part-*
```

The output matches the MapReduce results, validating correctness.

```
File  Edit  View  Terminal  Tabs  Help
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2026-01-28 01:01 /data
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /input
drwxr-xr-x   - hadoop supergroup          0 2026-02-10 14:52 /output
drwxr-xr-x   - hadoop supergroup          0 2026-02-18 18:20 /output_spark
drwx------   - hadoop supergroup          0 2026-02-10 14:44 /tmp
hadoop@yugal-Inspiron-3583:~$ hdfs dfs -cat /output_spark/part-*
(big,4)
(data,3)
(mapreduce,3)
(hadoop,4)
hadoop@yugal-Inspiron-3583:~$
```

Figure 50: Validating Spark output in HDFS

4.9 Step 9: Stop Spark and Hadoop Services

```
:quit
stop-yarn.sh
stop-dfs.sh
```

```
hadoop@yugal-Inspiron-3583:~$ stop-yarn.sh
stop-dfs.sh
Stopping nodemanagers
Stopping resourcemanager
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [yugal-Inspiron-3583]
hadoop@yugal-Inspiron-3583:~$
```

Figure 51: Stopping Spark session and Hadoop services

4.10 Step 10: Comparison with MapReduce

Feature	MapReduce	Spark
Processing Model	Disk-based	In-memory
Execution Speed	Slower	Faster
Code Complexity	More lines of code	Simple and concise
Iterative Processing	Inefficient	Efficient
Performance	Higher latency	Low latency

Table 1: Comparison between MapReduce and Spark

4.11 Result

The Spark application successfully processed the input dataset using in-memory computation. Word count results were generated and stored in HDFS, matching the output produced by the MapReduce implementation.

4.12 Conclusion

This assignment demonstrated in-memory analytics using Apache Spark. Compared to MapReduce, Spark required less code and executed faster due to memory-based processing. The experiment highlights Spark's suitability for high-performance analytics and iterative big data processing tasks.