

## Task-2

**Aim:** Implement the spark word count program on databricks and visualise how the RDD is partitioned between the executors and explore till the last thread in cluster.

### ❖ **Algorithm:**

- Start
- Create rdd using parallelize method (we can use various other methods also like `textFile`, `wholeTextFile` etc.)
- Convert the rdd into keyvalue pair using `map` (we can use various other method like `flatMap`, `wholeTextFile` etc.)
- Aggregate all the same key value using `reduceByKey`.
- Print the output using `collect()` method
- End.

### ❖ **Program of word count problem:**

```
data = [1,2,3,4,5]

rdd = sc.parallelize(data,5)

rdd.collect()

word_count = rdd.map(lambda x: (x,1))\

                .reduceByKey(lambda x,y:(x+y))

word_count.collect()
```

### ❖ **Approach / How to do this on databricks.**

Steps are as follow:

Step 1: Open the Databricks account

Step 2: Create a cluster and attach it to the notebook.

Step 3: Write the code of word count.

a) Initially I create a rdd(resilient distributed dataset) and set the number of partitions is 5.

The screenshot shows a Databricks notebook titled 'Task2 (Python)'. The code in the notebook is as follows:

```

1 from pyspark.sql import SparkSession
2 spark = SparkSession.builder\
3     .master("local")\
4     .appName("Task2")\
5     .getOrCreate()
6 sc = spark.sparkContext

```

Command 1 took 0.49 seconds -- by 500067177@stu.upes.ac.in at 5/26/2021, 5:03:03 PM on Mycluster

```

1 data = [1,2,3,4,5]
2 rdd = sc.parallelize(data,5)
3 rdd.collect()

```

Out[2]: [1, 2, 3, 4, 5]

Command 2 took 3.48 seconds -- by 500067177@stu.upes.ac.in at 5/26/2021, 5:03:03 PM on Mycluster

Shift+Enter to run

Because I set the partition value manually to 5 So it create 5 tasks in the backend and it will be handled by 5 executor As we know number of partitions = number of executors.

The screenshot shows the Databricks Clusters page for 'Mycluster'. The 'Executors' tab is selected, displaying a summary table and a list of executors.

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	3.4 KiB / 3.9 GiB	0.0 B	8	0	0	5	5	8 s (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	3.4 KiB / 3.9 GiB	0.0 B	8	0	0	5	5	8 s (0.0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram
driver	10.172.215.56:42009	Active	0	3.4 KiB / 3.9 GiB	0.0 B	8	0	0	5	5	8 s (0.0 ms)	0.0 B	0.0 B	0.0 B	<a href="#">Thread Dump</a>	<a href="#">Heap Histogram</a>

Showing 1 to 1 of 1 entries

Previous Next

b) With the help of **Map** I convert the rdds into key value pair rdd and after that using **reduceByKey** I aggregate all the values of same key.

**Task2 (Python)**

Mycluster | File | Edit | View: Standard | Permissions | Run All | Clear

Cmd 1

```
1 from pyspark.sql import SparkSession
2 spark = SparkSession.builder\
3     .master("local")\
4     .appName("Task2")\
5     .getOrCreate()
6 sc = spark.sparkContext
```

Command took 0.49 seconds -- by 500067177@stu.upes.ac.in at 5/26/2021, 5:03:03 PM on Mycluster

Cmd 2

```
1 data = [1,2,3,4,5]
2 rdd = sc.parallelize(data,5)
3 rdd.collect()
```

Out[2]: [1, 2, 3, 4, 5]

Command took 3.48 seconds -- by 500067177@stu.upes.ac.in at 5/26/2021, 5:03:03 PM on Mycluster

Cmd 3

```
1 word_count = rdd.map(lambda x: (x,1))\
2     .reduceByKey(lambda x,y:(x+y))
3 word_count.collect()
```

Out[3]: [(5, 1), (1, 1), (2, 1), (3, 1), (4, 1)]

Command took 2.48 seconds -- by 500067177@stu.upes.ac.in at 5/26/2021, 5:14:29 PM on Mycluster

Shift+Enter to run

Where, key represents the word and value represents the frequency of that word.

**Clusters / Mycluster**

Mycluster | Edit | Clone | Restart | Terminate | Delete

Configuration | Notebooks (1) | Libraries | Event Log | **Spark UI** | Driver Logs | Metrics | Apps | Spark Cluster UI - Master

Hostname: ec2-54-187-197-211.us-west-2.compute.amazonaws.com | Spark Version: 8.2.x-scala2.12

Jobs | Stages | Storage | Environment | **Executors** | SQL | JDBC/ODBC Server | Structured Streaming

**Executors**

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
<b>Active(1)</b>	0	15 KiB / 3.9 GiB	0.0 B	8	0	0	15	15	18 s (0.0 ms)	0.0 B	330 B	330 B	0
<b>Dead(0)</b>	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
<b>Total(1)</b>	0	15 KiB / 3.9 GiB	0.0 B	8	0	0	15	15	18 s (0.0 ms)	0.0 B	330 B	330 B	0

**Executors**

Show 20 entries

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump	Heap Histogram
driver	10.172.215.56:42009	Active	0	15 KiB / 3.9 GiB	0.0 B	8	0	0	15	15	18 s (0.0 ms)	0.0 B	330 B	330 B	<a href="#">Thread Dump</a>	<a href="#">Heap Histogram</a>

Showing 1 to 1 of 1 entries

[Previous](#) [Next](#)

After executing the map and reduceByKey the number of task increases from 5 to 15. The reason behind this is because each executor executes the map and reduceByKey for each partitioned value so it become 10 and added into the previously calculated value 5.

The screenshot shows the Databricks Spark UI for a cluster named 'Mycluster'. The 'Stages' tab is selected, displaying a table of completed stages. The table has columns for Stage ID, Pool Name, Description, Submitted, Duration, Tasks (Succeeded/Total), Input, Output, Shuffle Read, and Shuffle Write. There are three completed stages listed, each with a description of the operations performed (word\_count = rdd.map, collect, reduceByKey, and data = [1,2,3,4,5] rdd = sc.parallelize).

Stage ID	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	1632637924258362854	word_count = rdd.map(lambda x: (x,1))\n... collect at <command-3908961513749119>-3	2021/05/26 11:44:31	0.4 s	5/5			330.0 B	
1	1632637924258362854	word_count = rdd.map(lambda x: (x,1))\n... reduceByKey at <command-3908961513749119>-1	2021/05/26 11:44:29	2 s	5/5				330.0 B
0	1632637924258362854	data = [1,2,3,4,5] rdd = sc.parallelize(data,5)\n... collect at <command-3657528187293481>-3	2021/05/26 11:33:16	2 s	5/5				

By following these process we easily get our active executors.