ASSIGNMENT III

BAM-1043-01 Assignment
C#: c0932089_Assignment III
Name: Prabesh Rai
Date: 22 March, 2024

Question 01,
Why is Spark preferred over traditional MR jobs?
Ans:
Spark is preferred over traditional MapReduce (MR) jobs due to its enhanced usability and great performance. Firstly, Spark's in-memory computation model allows it to process data much faster than MapReduce by reducing the need for costly disk I/O operations. This speedup is particularly noticeable for iterative algorithms and interactive queries, where Spark's ability to cache data in memory provides a substantial advantage.

Moreover, Spark offers higher-level APIs in languages like Scala, Python, Java, and SQL, making it more accessible to a broader range of developers compared to the lengthy and complex MapReduce programming paradigm. Additionally, Spark's fault tolerance mechanisms, such as resilient distributed datasets (RDDs), ensure robustness against failures without sacrificing performance, which is a significant improvement over MapReduce's reliance on writing intermediate results to disk.

What is a SparkContext? Why is it needed?
Ans:
A SparkContext is an entry point to any Spark functionality in a Spark application. It represents the connection to a Spark cluster and is responsible for coordinating the execution of operations on that cluster.

SparkContext is needed for several reasons:

Cluster Management: SparkContext is responsible for managing the connections to the Spark cluster. It sets up the necessary configurations and resources required to interact with the cluster.

Resource Allocation: SparkContext allocates resources such as memory, CPU cores, and executors to different parts of the Spark application. It manages these resources efficiently based on the requirements of the tasks being executed.

Distributed Computing: SparkContext enables the distribution and execution of tasks across multiple nodes in the cluster. It breaks down the computation into smaller tasks and schedules them for execution on the cluster's worker nodes.

Fault Tolerance: SparkContext ensures fault tolerance by keeping track of the lineage of RDDs (Resilient Distributed Datasets) and DAGs (Directed Acyclic Graphs) of transformations and actions performed on the data. This lineage information allows Spark to recover from failures by recomputing lost data or tasks.

Interaction with Data Sources: SparkContext provides methods to interact with various data sources such as HDFS, S3, JDBC, etc. It allows Spark applications to read data from external sources, process it, and write the results back to the storage.

What is an RDD? Why is it used in Spark?
Ans:
RDD stands for Resilient Distributed Dataset. RDD is the fundamental data structure in Apache Spark and represents an immutable collection of objects that can be processed in parallel across a distributed cluster.

RDDs are used in Spark for several reasons:
Fault Tolerance: RDDs are fault-tolerant by design. They track the lineage of transformations applied to the base dataset, enabling the reconstruction of lost data partitions in case of failures. This fault tolerance mechanism ensures the reliability and robustness of Spark applications.

Parallel Processing: RDDs support parallel processing of data across distributed nodes in a cluster. They are partitioned into smaller chunks, and Spark processes these partitions in parallel, leveraging the distributed computing capabilities of the underlying cluster. This parallelism enables efficient utilization of cluster resources and faster data processing.

In-Memory Computation: RDDs can be cached in memory across the cluster nodes. This allows for faster data access and processing as compared to traditional disk-based processing frameworks like MapReduce. By caching intermediate results in memory, Spark minimizes the need for repetitive disk I/O operations, improving performance.

Lazy Evaluation: RDDs support lazy evaluation, meaning that transformations applied to RDDs are not computed immediately. Instead, they are evaluated only when an action is called. This optimization allows Spark to optimize the execution plan and perform transformations more efficiently.

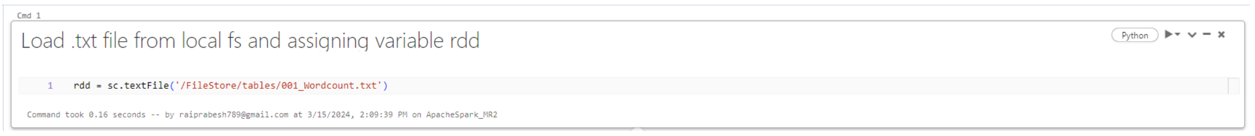
Versatility: RDDs offer a wide range of transformations and actions for data manipulation and processing. They support transformations like map, filter, reduceByKey, and actions like collect, count, saveAsTextFile, etc. This versatility enables users to perform various data processing tasks within a single framework.

What are Transformations? Which Transformations are used for this assignment?
Ans:
Transformations are operations applied to RDDs (Resilient Distributed Datasets) to create a new RDD. Transformations are lazy, meaning they do not compute the result immediately but instead build up a logical execution plan that gets executed when an action is called on the RDD. This lazy evaluation allows Spark to optimize the execution plan and improve performance.
Transformations like **map(func)** and **reduceByKey(func)** are used for this assignment.

What are Actions? Which Actions are used for this assignment?
Ans:
Actions are operations that trigger the execution of the logical plan created by transformations and return a result to the driver program or write data to external storage systems. Actions like **collect()** and **saveAsTextFile(path)** are used for this assignment.

Question 02,
Using Databricks notebook, please create a program for the MapReduce (for the given text file) using Apache Spark.

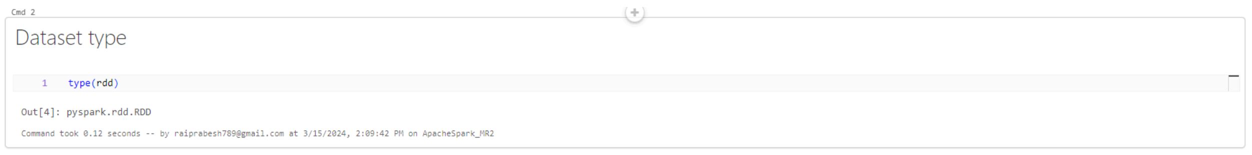
Screenshots of the programs for MapReduce using Apache Spark and Databricks are provided below:



Load .txt file from local fs and assigning variable rdd

```
rdd = sc.textFile('/FileStore/tables/001_Wordcount.txt')
```

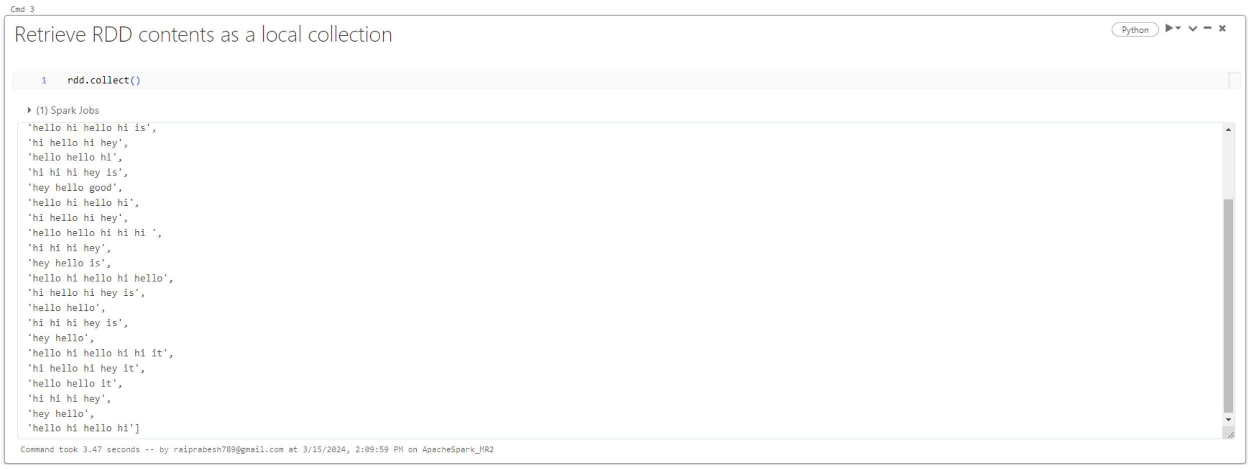
```
Command took 0.16 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:09:39 PM on ApacheSpark_MR2
```



Dataset type

```
type(rdd)
```

```
Out[4]: pyspark.rdd.RDD
Command took 0.12 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:09:42 PM on ApacheSpark_MR2
```


Cmd 3

Retrieve RDD contents as a local collection

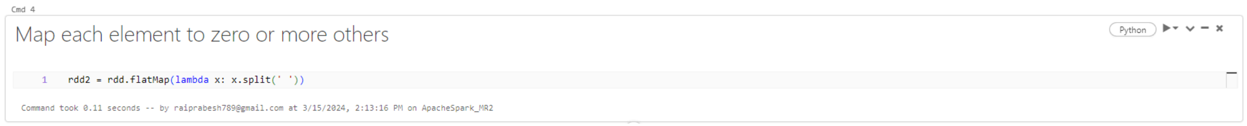
Python

```
rdd.collect()
```

(1) Spark Jobs

```
 'hello hi hello hi is',
 'hi hello hi hey',
 'hello hello hi',
 'hi hi hi hey is',
 'hey hello good',
 'hello hi hello hi',
 'hi hello hi hey',
 'hello hello hi hi hi ',
 'hi hi hi hey',
 'hey hello is',
 'hello hi hello hi hello',
 'hi hello hi hey is',
 'hello hello',
 'hi hi hi hey is',
 'hey hello',
 'hello hi hello hi hi it',
 'hi hello hi hey it',
 'hello hello it',
 'hi hi hi hey',
 'hey hello',
 'hello hi hello hi']
```

Command took 3.47 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:09:59 PM on ApacheSpark_MR2

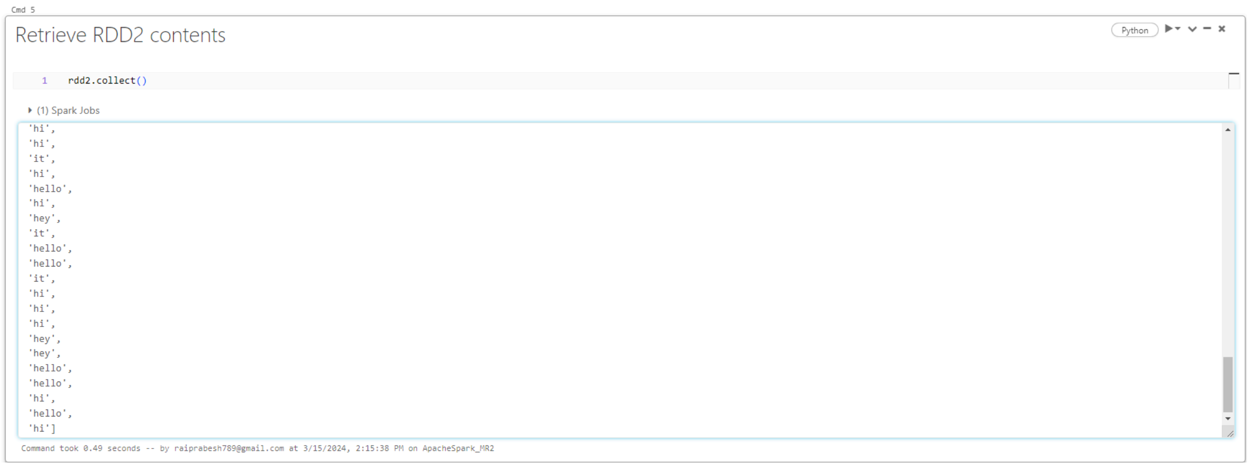

Cmd 4

Map each element to zero or more others

Python

```
rdd2 = rdd.flatMap(lambda x: x.split(' '))
```

Command took 0.11 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:13:16 PM on ApacheSpark_MR2


Cmd 5

Retrieve RDD2 contents

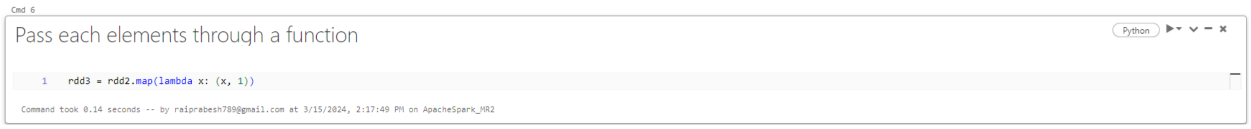
Python

```
rdd2.collect()
```

(1) Spark Jobs

```
 'hi',
 'hi',
 'it',
 'hi',
 'hello',
 'hi',
 'hey',
 'it',
 'hello',
 'hello',
 'it',
 'hi',
 'hi',
 'hi',
 'hey',
 'hey',
 'hello',
 'hello',
 'hi',
 'hello',
 'hi']
```

Command took 0.49 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:15:38 PM on ApacheSpark_MR2

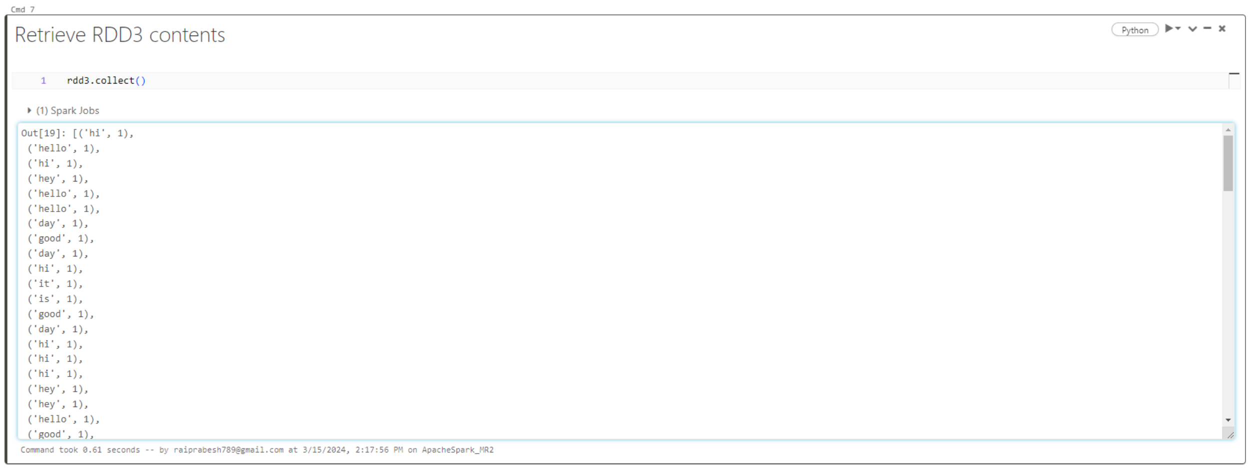

Cmd 6

Pass each elements through a function

Python

```
rdd3 = rdd2.map(lambda x: (x, 1))
```

Command took 0.14 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:17:49 PM on ApacheSpark_MR2


Cmd 7

Retrieve RDD3 contents

Python

```
rdd3.collect()
```

(1) Spark Jobs

```
Out[19]: [('hi', 1),
 ('hello', 1),
 ('hi', 1),
 ('hey', 1),
 ('hello', 1),
 ('hello', 1),
 ('day', 1),
 ('good', 1),
 ('day', 1),
 ('hi', 1),
 ('it', 1),
 ('is', 1),
 ('good', 1),
 ('day', 1),
 ('hi', 1),
 ('hi', 1),
 ('hi', 1),
 ('hey', 1),
 ('hey', 1),
 ('hello', 1),
 ('good', 1),
```

Command took 0.61 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:17:56 PM on ApacheSpark_MR2

Cmd 8

Aggregate data correcponding to a key with the help of an associative reduce function

```
rdd4 = rdd3.reduceByKey(lambda x, y: x+y)
```

Command took 0.12 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:20:58 PM on ApacheSpark_MR2

Cmd 9

Retrieve RDD4

Python

```
rdd4.collect()
```

▸ (1) Spark Jobs

```
Out[25]: [('good', 4),
 ('is', 6),
 ('', 1),
 ('hi', 41),
 ('hello', 31),
 ('hey', 15),
 ('day', 3),
 ('it', 4)]
```

Command took 1.03 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:21:01 PM on ApacheSpark_MR2

Cmd 10

Write elements to a text file

Python

```
rdd4.saveAsTextFile('dbfs:/results.txt')
```

▾ (1) Spark Jobs

▸ Job 14 View (Stages: 1/1, 1 skipped)

Command took 0.82 seconds -- by raiprabesh789@gmail.com at 3/15/2024, 2:40:53 PM on ApacheSpark_MR2