RDD Functions in Apache Spark - Manual with Examples

1. Creating an RDD

You can create an RDD from a collection or an external data source (like a text file).

```
# From a collection

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])


# From a text file

rdd = spark.sparkContext.textFile("path_to_file.txt")
```

2. Basic Transformations

Transformations create new RDDs from existing ones. These operations are lazy, meaning they are only executed when an action is called.

map: Applies a function to each element of the RDD and returns a new RDD.

```
rdd = spark.sparkContext.parallelize([1, 2, 3])

mapped_rdd = rdd.map(lambda x: x * 2)  # [2, 4, 6]
```

filter: Filters elements based on a condition.

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

filtered_rdd = rdd.filter(lambda x: x % 2 == 0)  # [2, 4]
```

flatMap: Similar to map, but flattens the result. Each element can be mapped to multiple elements (or none).

```
rdd = spark.sparkContext.parallelize(["Hello World", "Spark RDD"])

flat_mapped_rdd = rdd.flatMap(lambda x: x.split(" "))  # ['Hello', 'World', 'Spark', 'RDD']
```

distinct: Removes duplicate elements from the RDD.

```
rdd = spark.sparkContext.parallelize([1, 2, 2, 3, 3, 3])

distinct_rdd = rdd.distinct()  # [1, 2, 3]
```

union: Combines two RDDs into one.

```
rdd1 = spark.sparkContext.parallelize([1, 2])

rdd2 = spark.sparkContext.parallelize([3, 4])

union_rdd = rdd1.union(rdd2)  # [1, 2, 3, 4]
```

intersection: Returns elements common to both RDDs.

```
rdd1 = spark.sparkContext.parallelize([1, 2, 3])

rdd2 = spark.sparkContext.parallelize([2, 3, 4])

intersection_rdd = rdd1.intersection(rdd2)  # [2, 3]
```

join (for key-value RDDs): Performs an inner join on two key-value RDDs.

```
rdd1 = spark.sparkContext.parallelize([('a', 1), ('b', 2)])

rdd2 = spark.sparkContext.parallelize([('a', 3), ('b', 4)])
```

```
joined_rdd = rdd1.join(rdd2)  # [('a', (1, 3)), ('b', (2, 4))]
```

leftOuterJoin: Performs a left join, preserving all keys from the left RDD.

```
left_joined_rdd = rdd1.leftOuterJoin(rdd2)  # [('a', (1, 3)), ('b', (2, 4))]
```

3. Basic Actions

Actions trigger the execution of transformations and return results.

collect: Returns all elements of the RDD as a list (use carefully with large datasets).

```
rdd = spark.sparkContext.parallelize([1, 2, 3])

result = rdd.collect()  # [1, 2, 3]
```

take: Returns the first n elements from the RDD.

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

result = rdd.take(3)  # [1, 2, 3]
```

count: Counts the number of elements in the RDD.

```
rdd = spark.sparkContext.parallelize([1, 2, 3])

result = rdd.count()  # 3
```

first: Returns the first element of the RDD.

```
rdd = spark.sparkContext.parallelize([1, 2, 3])
result = rdd.first()  # 1
```

reduce: Aggregates the elements of the RDD using a function that takes two arguments and returns one.

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4])
result = rdd.reduce(lambda x, y: x + y)  # 10
```

4. Key-Value Pair Operations

These operations are specifically for RDDs of key-value pairs (tuples).

reduceByKey: Aggregates values for each key using a given associative function.

```
rdd = spark.sparkContext.parallelize([('a', 1), ('b', 2), ('a', 2)])
reduced_rdd = rdd.reduceByKey(lambda x, y: x + y)  # [('a', 3), ('b', 2)]
```

groupByKey: Groups the values for each key into an iterable collection.

```
rdd = spark.sparkContext.parallelize([('a', 1), ('b', 2), ('a', 2)])
grouped_rdd = rdd.groupByKey().mapValues(list)  # [('a', [1, 2]), ('b', [2])]
```

sortByKey: Sorts the RDD by keys.

```
rdd = spark.sparkContext.parallelize([('b', 2), ('a', 1), ('c', 3)])
```

```
sorted_rdd = rdd.sortByKey()  # [('a', 1), ('b', 2), ('c', 3)]
```

5. Persistence

Persist or cache the RDD in memory or disk for future reuse.

cache: Caches the RDD in memory.

```
rdd = spark.sparkContext.parallelize([1, 2, 3])
rdd.cache()
```

persist: Allows you to specify the storage level (e.g., memory, disk).

```
from pyspark import StorageLevel
rdd.persist(StorageLevel.MEMORY_AND_DISK)
```

6. Miscellaneous

Additional helpful RDD functions.

coalesce: Reduces the number of partitions in an RDD (useful for optimization).

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5], 4)
coalesced_rdd = rdd.coalesce(2)
```

repartition: Increases or decreases the number of partitions, potentially reshuffling the data.

```
rdd = spark.sparkContext.parallelize([1, 2, 3], 2)

repartitioned_rdd = rdd.repartition(4)
```