# PySpark RDD Interview Questions and Answers

## General Questions

**1. What is PySpark RDD?**
A PySpark RDD (Resilient Distributed Dataset) is the fundamental data structure in Apache Spark's Python API. It represents an immutable, fault-tolerant, and distributed collection of objects that can be processed in parallel across a cluster. Essentially, it's a collection of elements partitioned across the nodes in a cluster that can be operated on in parallel.

**2. Explain the key features of RDD in PySpark.**
* **Distributed:** Data is partitioned across multiple nodes in a cluster, allowing parallel processing.
* **Immutable:** Once an RDD is created, its contents cannot be changed. Transformations create new RDDs.
* **Resilient/Fault-Tolerant:** If a partition is lost due to a node failure, it can be automatically recomputed from its lineage.
* **Lazy Evaluation:** Transformations are not executed immediately but are recorded as a DAG of operations. Execution only happens when an action is called.
* **In-Memory Computing:** Spark tries to keep RDDs in memory for faster access.
* **Partitioned:** RDDs are divided into logical partitions, processed in parallel.

**3. What are the advantages of using RDDs in PySpark?**
* Fault Tolerance
* In-Memory Performance
* Flexibility (handles various data types)
* Low-Level Control
* Large-Scale Data Processing

**4. Describe the RDD lineage.**
RDD lineage is the DAG of transformations applied to a base RDD. Spark uses it to determine the execution plan and for fault recovery by recomputing lost partitions.

**5. How does PySpark handle fault tolerance in RDDs?**
Through RDD lineage. Lost partitions are recomputed by re-executing the transformations in the lineage.

**6. What are the different types of operations that can be performed on RDDs?**
* Transformations (lazy, create new RDDs)
* Actions (eager, trigger execution, return results)

**7. What is the difference between a transformation and an action in PySpark RDDs?**
* **Transformation:** Creates new RDD, lazy (e.g., `map()`, `filter()`).

* **Action:** Triggers execution, returns result, eager (e.g., `collect()`, `count()`).

**8. Explain lazy evaluation in the context of RDDs.**
Spark delays execution of transformations until an action is called, allowing for optimization of the execution plan.

**9. How can you create an RDD in PySpark?**
* Loading an external dataset (e.g., `sc.textFile()`).
* Parallelizing an existing collection (e.g., `sc.parallelize()`).

**10. What are the different methods to create an RDD in PySpark?**
* `sc.parallelize(collection)`
* `sc.textFile(path)`
* `sc.sequenceFile(path)`
* `sc.wholeTextFiles(path)`
* `sc.hadoopFile(path, input_format_class, key_class, value_class)`

**11. Explain the `map()` transformation with an example.**
Applies a function to each element, returning a new RDD.
```python
rdd = sc.parallelize([1, 2, 3]).map(lambda x: x * 2) # Output: [2, 4, 6]
```

**12. What is the difference between `map()` and `flatMap()` transformations?**

- `map()`: One-to-one mapping.
- `flatMap()`: One-to-many mapping, flattens the result.

**13. Describe the `filter()` transformation and its use case.** Returns a new RDD with elements for which the function returns `True`. Use case: selecting data based on a condition.

**14. What is the `reduce()` action and how does it work?** Aggregates all elements using an associative and commutative binary function.

**15. Explain the `collect()` action.** Returns all elements of the RDD as a list to the driver program (use with caution on large RDDs).

**16. What is the purpose of the `take()` action?** Returns the first `n` elements of the RDD as a list to the driver program.

**17. How does the `saveAsTextFile()` action work?** Writes the elements of the RDD to a text file (or directory of files) on a distributed file system.

**18. Describe the `count()` action.** Returns the number of elements in the RDD.

**19. What is the `union()` transformation in PySpark?** Returns a new RDD containing all elements from both RDDs (does not remove duplicates).

**20. Explain the `intersection()` transformation.** Returns a new RDD containing elements present in both RDDs (removes duplicates).

**21. Describe the `distinct()` transformation.** Returns a new RDD containing only the unique elements.

**22. How does the `groupByKey()` transformation work?** Groups values for each key into an iterable. Involves a shuffle.

**23. What is the difference between `groupByKey()` and `reduceByKey()`?**

- `groupByKey()`: Groups all values, full shuffle, less efficient for simple aggregations.
- `reduceByKey()`: Aggregates values per key, map-side aggregation, more efficient.

**24. Explain the `join()` transformation with an example.** Combines two RDDs of (key, value) pairs based on matching keys (inner join by default).

**25. What is the `cartesian()` transformation and when would you use it?** Returns all possible pairs of elements from two RDDs. Use with caution due to high cost.

**26. Describe the `coalesce()` transformation.** Reduces the number of partitions, tries to avoid full shuffle.

**27. What is the purpose of the `repartition()` transformation?** Redistributes data across a specified number of partitions (always involves a shuffle).

**28. Explain how to persist an RDD in memory.** Use `rdd.cache()` or `rdd.persist(StorageLevel.MEMORY_ONLY)`.

**29. What are the different storage levels available in PySpark for persisting RDDs?** `MEMORY_ONLY`, `MEMORY_AND_DISK`, `MEMORY_ONLY_SER`, `MEMORY_AND_DISK_SER`, `DISK_ONLY`, `OFF_HEAP`, with `_2` suffix for replication.

**30. How can you unpersist an RDD?** Use `rdd.unpersist()`.

**31. What is the `zip()` transformation in PySpark?** Combines two RDDs element-wise into pairs (requires same number of partitions and elements).

**32. Explain the `sample()` transformation.** Returns a sampled subset of the RDD (`withReplacement`, `fraction`, `seed`).

**33. Describe the `aggregate()` action and its use case.** Aggregates elements using `zeroValue`, `seqOp` (per partition), and `combOp` (combine partitions). Useful for complex aggregations.

**34. What is a broadcast variable and how is it used in PySpark?** Read-only shared variable cached on worker nodes, used for efficient sharing of large, read-only data.

**35. Explain the concept of accumulators in PySpark.** Shared variables that are only "added" to, used for efficient global aggregations (e.g., counters).

**36. What is the purpose of the `foreach()` action?** Applies a function to each element (for side effects).

**37. How can you debug a PySpark application?** Spark UI, logging, `print()` (executor logs), `collect()`, `take()`, `toDebugString()`, local mode, `explain()`.

**38. Explain the difference between `cache()` and `persist()`.** `cache()` is `persist(MEMORY_ONLY)`. `persist()` allows specifying different storage levels.

**39. What is the `glom()` transformation?** Returns an RDD of lists, where each list contains elements from one partition.

**40. Describe the `pipe()` transformation and its use case.** Pipes each partition through an external shell command. Useful for integrating external scripts.

**41. How does the `foreachPartition()` action work?** Applies a function to each partition (receives an iterator of elements). More efficient for per-partition setup.

**42. What is the significance of partitioning in RDDs?** Enables parallelism, data locality, performance optimization, and fault tolerance.

**43. Explain the `keyBy()` transformation.** Creates (key, value) pairs by applying a function to each element to generate the key.

**44. What are shuffle operations and why are they expensive?** Data redistribution across partitions (wide transformations). Expensive due to network I/O, disk I/O, serialization.

**45. How can you optimize PySpark RDD performance?** Minimize shuffles, optimize partitioning, use caching/persistence, broadcast variables, efficient transformations, etc.

**46. What is the `lookup()` transformation?** Action on (key, value) RDDs to return a list of values for a given key.

**47. Describe the `reduceByKeyLocally()` transformation.** Action that performs `reduceByKey` and returns a dictionary on the driver node (for small result sets).

**48. What is the `subtract()` transformation and when is it used?** Returns elements in the first RDD but not the second (set difference).

**49. Explain the `aggregateByKey()` transformation.** Aggregates values for each key independently using `zeroValue`, `seqOp`, and `combOp`.

**50. What are the best practices for writing PySpark RDD code?** Prefer DataFrames/Datasets, minimize shuffles, optimize partitioning, use caching, broadcast variables, avoid `collect()` on large data, use built-in functions, handle errors, manage resources, review execution plan.

## Scenario-Based Coding Questions

**1. Write a PySpark program to count the number of lines in a text file.**

```
Python
lines_rdd = sc.textFile("data/sample.txt")
line_count = lines_rdd.count()
```

**2. Given an RDD of numbers, write a PySpark program to compute the sum of all numbers.**

```
Python
numbers_rdd = sc.parallelize([1, 2, 3, 4, 5])
sum_of_numbers = numbers_rdd.reduce(lambda a, b: a + b)
```

**3. Create an RDD from a list of tuples and write a PySpark program to group the elements by key.**

```
Python
data_tuples = [("apple", 1), ("banana", 2), ("apple", 3)]
rdd = sc.parallelize(data_tuples)
grouped_rdd = rdd.groupByKey().mapValues(list)
```

**4. Write a PySpark program to find the maximum value in an RDD of integers.**

```
Python
numbers_rdd = sc.parallelize([10, 5, 20])
```

```
max_value = numbers_rdd.reduce(lambda a, b: max(a, b))
```

**5. Given an RDD of words, write a PySpark program to count the occurrences of each word.**

Python
```
words_rdd = sc.parallelize(["apple", "banana", "apple"])
word_counts = words_rdd.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
```

**6. Write a PySpark program to filter out even numbers from an RDD of integers.**

```python
numbers_rdd = sc.parallelize([1, 2, 3, 4, 5, 6])
even_numbers_rdd = numbers_rdd.filter(lambda x: x % 2 == 0)
```

**7. Create an RDD from a list of strings and write a PySpark program to convert all strings to uppercase.**

Python
```
strings_rdd = sc.parallelize(["hello", "world"])
uppercase_strings_rdd = strings_rdd.map(lambda s: s.upper())
```

**8. Write a PySpark program to join two RDDs based on a common key.**

Python
```
rdd1 = sc.parallelize([("id1", "A"), ("id2", "B")])
rdd2 = sc.parallelize([("id1", 1), ("id3", 2)])
joined_rdd = rdd1.join(rdd2)
```

**9. Given an RDD of (key, value) pairs, write a PySpark program to find the average value for each key.**

Python
```
data = [("K1", 10), ("K2", 20), ("K1", 30)]
rdd = sc.parallelize(data)
sum_count = rdd.aggregateByKey((0, 0), lambda acc, value: (acc[0] + value, acc[1] + 1),
lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1]))
average_rdd = sum_count.mapValues(lambda x: x[0] / x[1] if x[1] != 0 else 0)
```

**10. Write a PySpark program to find the distinct elements in an RDD.**

```python
rdd = sc.parallelize([1, 2, 2, 3, 1])
distinct_rdd = rdd.distinct()
```

**11. Create an RDD from a text file and write a PySpark program to find the top 10 most frequent words.**

```python
# Assuming 'file.txt' exists
words_rdd = sc.textFile("file.txt").flatMap(lambda line: line.lower().split()).map(lambda word:
(word, 1)).reduceByKey(lambda a, b: a + b)
top_10 = words_rdd.sortBy(lambda x: x[1], ascending=False).take(10)
```

**12. Write a PySpark program to perform a Cartesian product of two RDDs.**

```python
rdd1 = sc.parallelize([1, 2])
rdd2 = sc.parallelize(['a', 'b'])
cartesian_rdd = rdd1.cartesian(rdd2)
```

**13. Given an RDD of (key, value) pairs, write a PySpark program to sort the RDD by key.**

```python
data = [('c', 3), ('a', 1), ('b', 2)]
rdd = sc.parallelize(data)
sorted_rdd = rdd.sortByKey()
```

**14. Write a PySpark program to repartition an RDD into a specified number of partitions.**

```python
rdd = sc.parallelize(range(10), numPartitions=2)
repartitioned_rdd = rdd.repartition(4)
```

**15. Given an RDD of (key, value) pairs, write a PySpark program to perform a reduceByKey operation.**

```python
data = [('k1', 1), ('k2', 2), ('k1', 3)]
rdd = sc.parallelize(data)
reduced_rdd = rdd.reduceByKey(lambda a, b: a + b)
```

**16. Write a PySpark program to compute the average of an RDD of numbers.**

```python
numbers_rdd = sc.parallelize([1, 2, 3, 4, 5])
sum_val = numbers_rdd.sum()
count_val = numbers_rdd.count()
average = sum_val / count_val if count_val > 0 else 0
```

**17. Given an RDD of (key, value) pairs, write a PySpark program to combine values with the same key using a provided function.**

```python
data = [('k1', [1]), ('k2', [2]), ('k1', [3])]
rdd = sc.parallelize(data)
combined_rdd = rdd.reduceByKey(lambda a, b: a + b)
```

**18. Write a PySpark program to persist an RDD in memory and disk.**

```python
rdd = sc.parallelize(range(100))
rdd.persist(StorageLevel.MEMORY_AND_DISK)
```

**19. Given an RDD of (key, value) pairs, write a PySpark program to count the number of unique keys.**

```python
data = [('a', 1), ('b', 2), ('a', 3)]
rdd = sc.parallelize(data)
unique_keys_count = rdd.keys().distinct().count()
```

**20. Write a PySpark program to sample 10% of an RDD.**

```python
rdd = sc.parallelize(range(100))
sampled_rdd = rdd.sample(False, 0.1)
```

**21. Given an RDD of sentences, write a PySpark program to count the number of words in each sentence.**

```python
sentences = sc.parallelize(["hello world", "spark rdd"])
```

```
word_counts = sentences.map(lambda s: (s, len(s.split())))
```

**22. Write a PySpark program to compute the average length of words in an RDD.**

Python
```
words = sc.parallelize(["hello", "world"])
total_length = words.map(len).sum()
total_words = words.count()
average_length = total_length / total_words if total_words > 0 else 0
```

**23. Given an RDD of (key, value) pairs, write a PySpark program to filter out keys with values less than a specified threshold.**

Python
```
data = [('a', 5), ('b', 10), ('c', 3)]
rdd = sc.parallelize(data)
threshold = 7
filtered_rdd = rdd.filter(lambda kv: kv[1] >= threshold)
```

**24. Write a PySpark program to merge two RDDs into one.**

Python
```
rdd1 = sc.parallelize([1, 2])
rdd2 = sc.parallelize([3, 4])
merged_rdd = rdd1.union(rdd2)
```

**25. Given an RDD of integers, write a PySpark program to find the second largest number.**

Python
```
numbers = sc.parallelize([1, 5, 2, 8])
second_largest = numbers.distinct().sortBy(lambda x: x, ascending=False).take(2)[1] if
numbers.distinct().count() >= 2 else None
```

**26. Write a PySpark program to count the number of partitions in an RDD.**

Python
```
rdd = sc.parallelize(range(10), numPartitions=3)
num_partitions = rdd.getNumPartitions()
```

**27. Given an RDD of (key, value) pairs, write a PySpark program to compute the sum of values for each key.**

```python
data = [('k1', 1), ('k2', 2), ('k1', 3)]
rdd = sc.parallelize(data)
sum_by_key = rdd.reduceByKey(lambda a, b: a + b)
```

**28. Write a PySpark program to perform a union of two RDDs.**

```python
rdd1 = sc.parallelize([1])
rdd2 = sc.parallelize([2])
union_rdd = rdd1.union(rdd2)
```

**29. Given an RDD of (key, value) pairs, write a PySpark program to find the key with the maximum value.**

```python
data = [('a', 10), ('b', 5)]
rdd = sc.parallelize(data)
max_key = rdd.reduce(lambda a, b: a if a[1] > b[1] else b)[0] if rdd.count() > 0 else None
```

**30. Write a PySpark program to create an RDD from a list of numbers and compute their square root.**

```python
import math
numbers = sc.parallelize([4, 9, 16])
sqrt_rdd = numbers.map(math.sqrt)
```

**31. Given an RDD of strings, write a PySpark program to remove duplicate strings.**

```python
strings = sc.parallelize(["a", "b", "a"])
unique_strings = strings.distinct()
```

**32. Write a PySpark program to sort an RDD of numbers in descending order.**

```python
numbers = sc.parallelize([3, 1, 4, 2])
```

```
sorted_desc = numbers.sortBy(lambda x: x, ascending=False)
```

**33. Given an RDD of (key, value) pairs, write a PySpark program to create a new RDD with the values incremented by 1 for each key.**

```python
data = [('k1', 1), ('k2', 2)]
rdd = sc.parallelize(data)
incremented_values = rdd.mapValues(lambda v: v + 1)
```

**34. Write a PySpark program to coalesce an RDD into a specified number of partitions.**

```python
rdd = sc.parallelize(range(10), numPartitions=4)
coalesced_rdd = rdd.coalesce(2)
```

**35. Given an RDD of (key, value) pairs, write a PySpark program to find the average value per partition.**

```python
rdd = sc.parallelize([(1, 10), (1, 20), (2, 30)], numPartitions=2)
avg_per_partition = rdd.mapPartitions(lambda it: [sum(v for k, v in it) / len(list(it)) if list(it) else 0])
```

**36. Write a PySpark program to compute the sum of squares of an RDD of numbers.**

```python
numbers = sc.parallelize([1, 2, 3])
sum_of_squares = numbers.map(lambda x: x**2).reduce(lambda a, b: a + b)
```

**37. Given an RDD of (key, value) pairs, write a PySpark program to filter keys based on a specified prefix.**

```python
data = [('apple', 1), ('apricot', 2), ('banana', 3)]
rdd = sc.parallelize(data)
filtered_keys = rdd.filter(lambda kv: kv[0].startswith('ap'))
```

**38. Write a PySpark program to aggregate elements of an RDD using a provided associative function and a neutral zero value.**

```python
numbers = sc.parallelize([1, 2, 3])
aggregated = numbers.aggregate(0, lambda acc, x: acc + x, lambda a, b: a + b)
```

**39. Given an RDD of (key, value) pairs, write a PySpark program to find the minimum value for each key.**

```python
data = [('k1', 5), ('k2', 10), ('k1', 2)]
rdd = sc.parallelize(data)
min_by_key = rdd.reduceByKey(lambda a, b: min(a, b))
```

**40. Write a PySpark program to zip two RDDs together.**

```python
rdd1 = sc.parallelize(['a', 'b'])
rdd2 = sc.parallelize([1, 2])
zipped_rdd = rdd1.zip(rdd2)
```

**41. Given an RDD of sentences, write a PySpark program to create an RDD of words.**

```python
sentences = sc.parallelize(["hello world"])
words_rdd = sentences.flatMap(lambda s: s.split())
```

**42. Write a PySpark program to count the occurrences of each character in a text file.**

```python
# Assuming 'chars.txt' exists
char_counts = sc.textFile("chars.txt").flatMap(list).map(lambda char: (char,
1)).reduceByKey(lambda a, b: a + b)
```

**43. Given an RDD of (key, value) pairs, write a PySpark program to filter out keys that have values in a specified range.**

```python
data = [('a', 5), ('b', 15), ('c', 10)]
rdd = sc.parallelize(data)
filtered_range = rdd.filter(lambda kv: 8 <= kv[1] <= 12)
```

**44. Write a PySpark program to convert an RDD of strings to an RDD of (string, length) pairs.**

```python
strings = sc.parallelize(["one", "two"])
length_rdd = strings.map(lambda s: (s, len(s)))
```

**45. Given an RDD of (key, value) pairs, write a PySpark program to group the keys and collect the values into a list.**

```python
data = [('k1', 'v1'), ('k2', 'v2'), ('k1', 'v3')]
rdd = sc.parallelize(data)
grouped_values = rdd.groupByKey().mapValues(list)
```

**46. Write a PySpark program to find the median value of an RDD of numbers.**

```python
numbers = sc.parallelize([3, 1, 4, 2, 5])
sorted_nums = sorted(numbers.collect())
n = len(sorted_nums)
median = sorted_nums[n // 2] if n % 2 == 1 else (sorted_nums[n // 2 - 1] + sorted_nums[n // 2]) /
2 if n > 0 else None
```

**47. Given an RDD of (key, value) pairs, write a PySpark program to perform a left outer join with another RDD.**

```python
rdd1 = sc.parallelize([('a', 1), ('b', 2)])
rdd2 = sc.parallelize([('a', 'x'), ('c', 'y')])
left_join = rdd1.leftOuterJoin(rdd2)
```

**48. Write a PySpark program to compute the standard deviation of an RDD of numbers.**

```python
numbers = sc.parallelize([1, 2, 3, 4, 5])
n = numbers.count()
mean = numbers.sum() / n if n > 0 else 0
std_dev = math.sqrt(numbers.map(lambda x: (x - mean)**2).sum() / (n - 1)) if n > 1 else 0
```

**49. Given an RDD of (key, value) pairs, write a PySpark program to count the number of keys that start with a specified letter.**

```python
data = [('apple', 1), ('banana', 2), ('ant', 3)]
rdd = sc.parallelize(data)
count_starting_with_a = rdd.filter(lambda kv: kv[0].startswith('a')).keys().distinct().count()
```

**50. Write a PySpark program to combine two RDDs of (key, value) pairs into a single RDD.**

```python
rdd1 = sc.parallelize([('k1', 1)])
rdd2 = sc.parallelize([('k2', 2)])
combined_rdd = rdd1.union(rdd2)
```

```python
# Final cleanup
spark.stop()
```