**Accumulators**

* **Definition:** Variables that are “added” to through an associative and commutative operation.
* **Use Case:** Useful for aggregating statistics across multiple tasks.
* **Example:**

# Define an accumulator variable

acc = spark.sparkContext.accumulator(0)

# Use in RDD transformations

def count\_lines(line):

global acc

if "error" in line:

acc.add(1)

return line

rdd = spark.sparkContext.textFile("logfile.txt")

rdd.foreach(count\_lines)

print("Error count:", acc.value)

**Broadcast Variables**

* **Definition:** Variables that are cached on each machine rather than shipped with tasks.
* **Use Case:** Efficiently share large read-only data between tasks.
* **Example:**

# Broadcast a large lookup table

broadcast\_var = spark.sparkContext.broadcast({'key1': 'value1', 'key2': 'value2'})

def lookup\_value(key):

return broadcast\_var.value.get(key, 'default')

rdd = spark.sparkContext.parallelize(['key1', 'key3'])

rdd.map(lookup\_value).collect()

**Piping to External Programs**

* **Definition:** Allows Spark to work with external tools via pipes.
* **Use Case:** Integrate with other systems or custom scripts.
* **Example:**

# Piping data to an external Python script

rdd = spark.sparkContext.textFile("data.txt")

rdd.pipe("python external\_script.py").collect()

**Numeric RDD Operations**

* **Definition:** Operations performed on RDDs containing numeric data.
* **Example:**

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

sum\_rdd = rdd.reduce(lambda x, y: x + y) # Sum

mean\_rdd = rdd.mean() # Mean (if using DataFrame API)

**Spark Runtime Architecture**

* **Components:**
  + **Driver Program:** Coordinates the execution of tasks.
  + **Cluster Manager:** Manages resources across the cluster.
  + **Worker Nodes:** Executes tasks and stores data.

**Deploying Applications**

* **Modes:**
  + **Local Mode:** For development/testing on a single machine.
  + **Standalone Mode:** Spark’s built-in cluster manager.
  + **YARN/Mesos:** External cluster managers.
* **Example:**

# Running Spark in standalone mode

spark-submit --master spark://master:7077 app.py

**Functional Programming: Lambda in Spark**

* **Definition:** Using anonymous functions for transformations.
* **Use Case:** Apply custom logic without defining named functions.
* **Example:**

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

squared\_rdd = rdd.map(lambda x: x \*\* 2)

**Map, FlatMap, Filter, and Sort**

* **Map:** Applies a function to each element.
* **FlatMap:** Applies a function and flattens the result.
* **Filter:** Filters elements based on a condition.
* **Sort:** Sorts elements.
* **Example:**

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

map\_rdd = rdd.map(lambda x: x + 1)

flatmap\_rdd = rdd.flatMap(lambda x: [x, x \* 2])

filter\_rdd = rdd.filter(lambda x: x > 1)

sorted\_rdd = rdd.sortBy(lambda x: x, ascending=False)

**Actions**

* **Definition:** Operations that trigger computation and return results.
* **Examples:**
  + **collect():** Returns all elements.
  + **count():** Counts the number of elements.
  + **take(n):** Returns the first n elements.
  + **saveAsTextFile(path):** Saves the RDD to a file.
* **Example:**

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

count = rdd.count()

first\_elements = rdd.take(2)

**Partition Operations: MapPartitions and PartitionBy**

* **MapPartitions:** Apply a function to each partition.
* **PartitionBy:** Re-partitions RDD based on a partitioning function.
* **Example:**

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

partitioned\_rdd = rdd.mapPartitions(lambda iter: [sum(iter)])

**Set Operations: Join, Union, Full Right, Left Outer, and Cartesian**

* **Join:** Combines RDDs based on keys.
* **Union:** Combines two RDDs.
* **Full Right/Left Outer Join:** Includes all keys from one or both RDDs.
* **Cartesian:** Computes the Cartesian product.
* **Example:**

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

rdd2 = spark.sparkContext.parallelize([(1, 'x'), (3, 'y')])

joined\_rdd = rdd1.join(rdd2) # Inner Join

**Combining, Aggregating, Reducing, and Grouping on PairRDDs**

* **Combining:** Merging values for the same key.
* **Aggregating:** Performing aggregation operations.
* **Reducing:** Aggregating values across the entire RDD.
* **Grouping:** Grouping elements by keys.
* **Example:**

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

aggregated\_rdd = rdd.reduceByKey(lambda x, y: x + y)

**ReduceByKey vs. GroupByKey**

* **ReduceByKey:** Combines values with the same key before shuffling data.
* **GroupByKey:** Groups values with the same key and then combines.
* **Comparison:**
  + **reduceByKey** is more efficient for large datasets.
* **Example:**

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

reduce\_by\_key\_rdd = rdd.reduceByKey(lambda x, y: x + y)

**Grouping Data into Buckets with Histogram**

* **Definition:** Creating histograms to categorize data into ranges.
* **Use Case:** Analyze distribution of numeric data.
* **Example:**

import numpy as np

data = [1, 2, 2, 3, 4, 5, 6, 7, 8]

histogram, bins = np.histogram(data, bins=4)