NAME: Vedashree Bhalerao

CLASS: AIA-1

BATCH: B

ROLL NO: 2213191

Big Data Analytics Experiment no. 03

Aim: a) To Install Apache Spark.

b) write a program in Spark for Word Count with Unit Tests: Change Application

Theory:

What is Apache Spark?

Apache Spark is an open-source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark is designed to be fast for both interactive queries and large-scale batch processing. It extends the Map Reduce model to efficiently support more types of computations, including interactive queries and stream processing.

Why Use Apache Spark?

Speed: Spark's in-memory computing capabilities make it up to 100 times faster than Hadoop MapReduce.

Ease of Use: Spark has easy-to-use APIs for operating on large datasets, which include a comprehensive set of libraries.

General-purpose: It supports various programming languages such as Java, Scala, Python, and R, making it a versatile tool.

Unified Engine: Spark provides a unified engine that can handle various types of data processing workloads, including batch processing, stream processing, and machine learning.

Prerequisites:

1. Java Development Kit (JDK): Apache Spark requires Java to be installed on your system. Ensure you have JDK 8 or later.

2. Scala: Apache Spark is written in Scala, and it's useful to have Scala installed if you want to use the Scala API.

3. Python: For using the PySpark API, ensure Python 2.7 or 3.4 and later versions are installed.

Steps to Install Apache Spark:

Step 1: Install Java Development Kit (JDK)

1.Download JDK: Go to the [Oracle JDK download page](https://www.oracle.com/java/technologies/javase-downloads.html) or use OpenJDK.

2. Install JDK:

   - On Windows: Run the installer and follow the setup instructions.

   - On macOS: Use Homebrew with the command `brew install openjdk`.

   - On Linux: Use your package manager, e.g., `sudo apt-get install openjdk-8-jdk`.

3. Set JAVA_HOME:

   - On Windows: Add the JDK bin directory to the system PATH.

   - On macOS/Linux: Add the following lines to your `.bashrc` or `.zshrc`:

```bash
export JAVA_HOME=/path/to/your/jdk

export PATH=$JAVA_HOME/bin:$PATH
```

Step 2: Download Apache Spark

1. Go to the Spark Download Page**: Visit the [Apache Spark download page](https://spark.apache.org/downloads.html).

2. Select Spark Release: Choose a Spark release version (e.g., 3.1.2) and select a pre-built package for Hadoop (e.g., Pre-built for Apache Hadoop 2.7 and later).

3. Download Spark: Download the selected Spark package and extract it to a directory of your choice.

Step 3: Set Up Environment Variables

1. Set SPARK_HOME:

   - On Windows: Add `SPARK_HOME` environment variable pointing to your Spark directory.

   - On macOS/Linux: Add the following lines to your `.bashrc` or `.zshrc`:

```bash
export SPARK_HOME=/path/to/spark
export PATH=$SPARK_HOME/bin:$PATH
```

2. Configure Spark:

   - Copy the template configuration file:

```bash
cp $SPARK_HOME/conf/spark-env.sh.template $SPARK_HOME/conf/spark-env.sh
```

   - Edit `spark-env.sh` to configure your Spark settings, such as setting `SPARK_MASTER_HOST`.

Step 4: Verify Installation

1. Start Spark Shell:

   - Open a terminal and run the following command:

```bash
spark-shell
```

   - You should see the Spark shell starting, and you can run Spark commands in Scala.

2. Start PySpark:

- For Python API, run:
  ```bash
  pyspark
  ```
- We should see the PySpark shell starting, allowing you to run Spark commands in Python.

b) write a program in Spark for Word Count with Unit Tests: Change Application

Theory

What is Word Count?

Word Count is a simple yet fundamental algorithm used to demonstrate the capabilities of data processing frameworks like Apache Spark. The algorithm reads a text file, splits it into words, and counts the occurrences of each word.

Why Use Spark for Word Count?

Scalability: Spark can handle large datasets distributed across a cluster of machines.

Speed: Spark's in-memory computing capabilities make it faster than traditional disk-based processing.

Ease of Use: Spark provides high-level APIs in Java, Scala, Python, and R, making it easy to implement common data processing tasks.

Spark Programming Model

Spark's programming model is based on Resilient Distributed Datasets (RDDs), which are fault-tolerant collections of elements that can be operated on in parallel. The main operations on RDDs are transformations (e.g., map, filter, reduceByKey) and actions (e.g., collect, saveAsTextFile).

Steps to Implement Word Count in Spark

Initialize Spark Context: Set up the environment to interact with Spark.

- Read Input Data: Load the text file into an RDD.
- Transform Data: Split the lines into words and map each word to a key-value pair (word, 1).
- Aggregate Data: Reduce the key-value pairs by key to get the word counts.
- Save or Display Output: Save the results to an output file or display them on the console.

Unit Testing in Spark

Unit tests are essential to ensure the correctness of your Spark application. You can use testing frameworks like unittest in Python or ScalaTest in Scala to write unit tests for your Spark application. Key aspects to test include:

Correctness of the transformation logic.

Handling of edge cases (e.g., empty lines, special characters).

Performance and scalability.

Algorithm

Word Count Algorithm in Spark

1. Initialize Spark Context:

   Create a SparkContext object to interact with Spark.

2. Read Input Data:

   Use the textFile method to load the input file into an RDD.

3. Transform Data:

   Use flatMap to split each line into words.

   Use map to create key-value pairs (word, 1) for each word.

4. Aggregate Data:

Use reduceByKey to sum the counts for each word.

5. Save or Display Output:

Use saveAsTextFile or collect to output the results.

Conclusion:

We have successfully installed the Apache Spark, creating a basic word count program in Spark, and adding unit tests to ensure the correctness of the application. With these steps, we can leverage Spark's powerful capabilities for data processing and ensure the reliability of our applications through comprehensive testing. By integrating unit tests, one can maintain code quality and quickly identify any issues, making your Spark applications robust and dependable.

CODE:

```
def word_count(s):
    """
    This function takes a string as input and returns the number of
words in the string.
    """
    words = s.split()
    return len(words)

import unittest

class TestWordCount(unittest.TestCase):
    def test_empty_string(self):
        self.assertEqual(word_count(""), 0)

    def test_single_word(self):
        self.assertEqual(word_count("Hello"), 1)

    def test_multiple_words(self):
        self.assertEqual(word_count("Hello world"), 2)

    def test_leading_and_trailing_spaces(self):
        self.assertEqual(word_count("   Hello world   "), 2)

    def test_multiple_spaces_between_words(self):
        self.assertEqual(word_count("Hello     world"), 2)

    def test_newline_characters(self):
```

```python
        self.assertEqual(word_count("Hello\nworld"), 2)

    def test_tab_characters(self):
        self.assertEqual(word_count("Hello\tworld"), 2)

# Run the tests
unittest.main(argv=[''], verbosity=2, exit=False)
```

OUTPUT:

```
test_empty_string (__main__.TestWordCount) ... ok
test_leading_and_trailing_spaces (__main__.TestWordCount) ... ok
test_multiple_spaces_between_words (__main__.TestWordCount) ... ok
test_multiple_words (__main__.TestWordCount) ... ok
test_newline_characters (__main__.TestWordCount) ... ok
test_single_word (__main__.TestWordCount) ... ok
test_tab_characters (__main__.TestWordCount) ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.030s

OK
<unittest.main.TestProgram at 0x79be6818be80>
```