MapReduce is a programming model designed for processing and generating large datasets in parallel across a distributed cluster of computers. It simplifies the complexities of parallel programming by dividing tasks into two main functions: **Map** and **Reduce**. This model is foundational in big data processing and is utilized in various applications across different industries

Apache Hadoop and Apache Spark are both powerful open-source frameworks for processing large datasets, but they differ in architecture, performance, and use cases.

1. Processing Models:

- **Hadoop:** Utilizes the MapReduce programming model, processing data in discrete batches. Each MapReduce job reads data from the disk, processes it, and writes the results back to the disk, which can introduce latency due to frequent disk I/O operations.

- **Spark:** Employs a Directed Acyclic Graph (DAG) execution engine that performs computations in memory, significantly reducing disk I/O. This in-memory processing enables Spark to handle iterative and interactive tasks more efficiently.

Step1

sudo apt update

sudo apt install openjdk-11-jdk

java -version

step 2

```
sudo apt install scala
scala -version
```

```
Scala code runner version 2.11.12 --
Copyright 2002-2017, LAMP/EPFL
```

step 3

```
echo "deb
https://repo.scala-sbt.org/scalasbt/debian all
main" | sudo tee
/etc/apt/sources.list.d/sbt.list
curl -sL
"https://keyserver.ubuntu.com/pks/lookup?o
p=get&search=0x99e82a75642ac823" | sudo
apt-key add
sudo apt update
```

step 4

wget https://dlcdn.apache.org/spark/spark-3.5.5/spark-3.5.5-bin-hadoop3.tgz

step 5

tar -xvzf spark-3.5.5-bin-hadoop3.tgz

mv spark-3.5.5-bin-hadoop3 ~/spark

step 6

export SPARK_HOME=~/spark

export PATH=$SPARK_HOME/bin:$PATH

comment

step 7:

```
mkdir spark-scala-app
cd spark-scala-app
```

```
mkdir -p src/main/scala
touch build.sbt
touch src/main/scala/WordCount.scala
```

```
spark-shell
```

```
scala> val text = sc.parallelize(Seq("spark is fast",
"scala is powerful", "spark scala together"))
```

`text` is an RDD of strings
RDD stands for Resilient Distributed Dataset.

It is the core data structure in Apache Spark.

```
text: org.apache.spark.rdd.RDD[String] =
ParallelCollectionRDD[0] at parallelize at
<console>:23



scala>



scala> val words = text.flatMap(_.split("\\s+"))

words: org.apache.spark.rdd.RDD[String] =
MapPartitionsRDD[1] at flatMap at <console>:23
```

`words` is an RDD of individual words
The key part is the **regular expression** `\\s+`.

`\s` (single backslash) means: **any whitespace character**.
It can match:

- space `" "`
- tab `\t`
- newline `\n`
- carriage return `\r`, etc.

In Scala, we write `\\s` instead of `\s` because:

- In a string literal, `\` is an **escape character**.
- So to get a literal `\`, you write it as `\\`.

"\\s" → means regex `\s` → matches any whitespace

🔹 `+` – What does it mean?

- `+` in regex means: **"one or more times"**

So:

```scala
"\\s+"  → matches **one or more whitespace characters**
```

`"hello   world"` → split into `"hello"` and `"world"` (because of 3 spaces)

`"one\ttwo\nthree"` → split into `"one"`, `"two"`, `"three"`
👉 splits each line into words, no matter how many spaces or tabs separate them.

🔹 What is Whitespace?

Whitespace means **any character that makes space** on the screen but is **not visible**.
They are used to **separate words or lines**, but they don't display actual symbols or letters.

🔸 **Common Whitespace Characters:**

Character	Description	Looks Like
`" "` (space)	Regular space	␣ (invisible)
`\t`	Tab	→ (moves cursor to next tab stop)
`\n`	Newline (line break)	↓ (moves to next line)
`\r`	Carriage return	⏎ (used in old systems)
`\f`	Form feed	— (rarely used now)

```
"hello world"      → space
"hello\tworld"     → tab
"hello\nworld"      → newline
```

scala> val wordPairs = words.map(word => (word, 1))

wordPairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[2] at map at <console>:23

`wordPairs` is an RDD of (String, Int) tuples

scala> val wordCounts = wordPairs.reduceByKey(_ + _)

wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[3] at reduceByKey at <console>:23

This line is used to **count how many times each word appears**. Here's how it works step by step:

("spark", 1), ("is", 1), ("fast", 1), ("spark", 1), ("is", 1), ...

This means:

- Group all the key-value pairs **by key** (i.e., by word).
- For each group, **add up** the values (the 1s).

🔸 **`_ + _` Explanation:**

This is a **shorthand** for:

scala

CopyEdit

```
(x, y) => x + y
```

```
("spark", 1), ("spark", 1) →
("spark", 2)
```

```
scala>


scala> wordCounts.collect().foreach(println)
(scala,2)
(together,1)
(powerful,1)
(is,2)
(fast,1)
(spark,2)
```