**All About Spark- Jobs, Stages and Tasks**

Introduction

The amount of data evolving daily is increasing, and so are the techniques to handle it. Processing this large amount of data to get valuable insights is challenging, but we have come a long way in advancing techniques to process and handle this large amount of data.

One of the big data processing engines is Apache Spark. It not only computes data parallelly but is also designed to run on a cluster of engines(distributed in nature). Its ability to process big data efficiently has even outshined Hadoop [MapReduce](https://www.analyticsvidhya.com/blog/2022/09/frequent-itemset-mining-using-mapreduce-on-hadoop/).

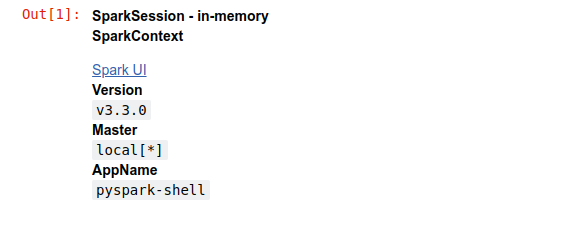
Now let’s dive into how Apache Spark executes code internally.

We are familiar with these lines of code below, but it is quite interesting to note how these lines get executed internally.

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

spark



sc = spark.sparkContext

# Parallelize rdd

rdd\_0 = sc.parallelize(["She saw Sharif's shoes on the sofa but was she so sure those were Sharif's shoes she saw?"])

# Tokenize sentence into words

rdd\_1 = rdd\_0.flatMap(lambda x: x.split())

# Create pair rdd

rdd\_2 = rdd\_1.map(lambda x: (x, 1))

# Count occurences of words

rdd\_3 = rdd\_2.reduceByKey(lambda a,b: a+b)

# Persist rdd

rdd\_3.persist()

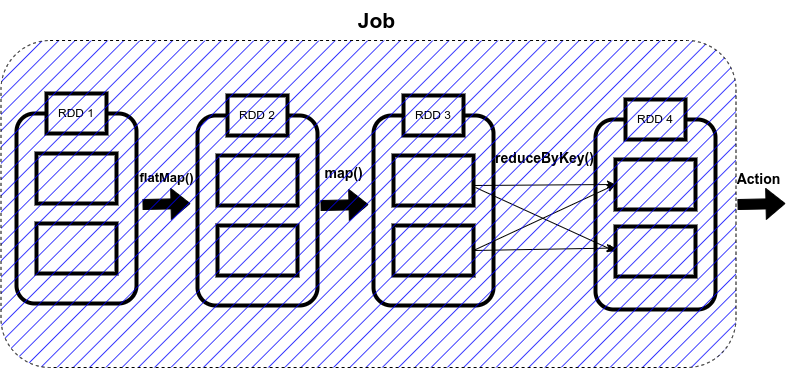
# Collect rdd

rdd\_3.collect()

In the above code, we have a Spark application with multiple RDD transformations executing on a dataset, and at the end, we are computing the result with a single action function.

Spark Jobs

So, what Spark does is that as soon as action operations like collect(), count(), etc., is triggered, the driver program, which is responsible for launching the spark application as well as considered the entry point of any spark application, converts this spark application into a single job which can be seen in the figure below.



A job is defined as a series of stages combined.

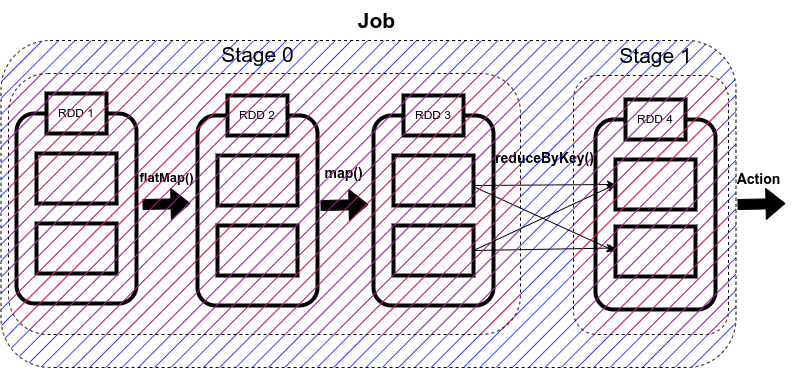
Spark Stages

Now here comes the concept of Stage. Whenever there is a shuffling of data over the network, Spark divides the job into multiple stages. Therefore, a stage is created when the shuffling of data takes place.

These stages can be either processed parallelly or sequentially depending upon the dependencies of these stages between each other. If there are two stages, Stage 0 and Stage 1, and if they are not sequentially dependent, they will be executed parallelly.

The sequential processing of RDDs in a single stage is called pipelining.

So, in our code, we have used reduceByKey() function, which shuffles our data in order to group the same keys. Since shuffling of data is taking place only once, our job will be divided into two stages as shown in the figure below.



There are two types of stages in Spark:

1.ShuffleMapStage in Spark

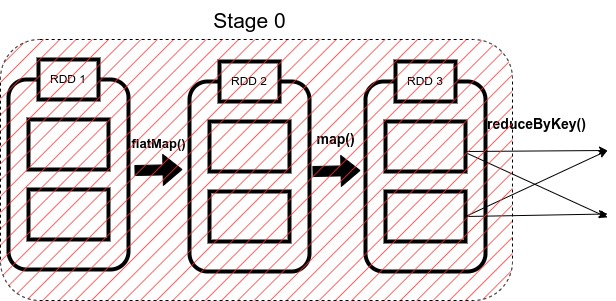
2. ResultStage in Spark

1. ShuffleMapStage

As the name suggests, it is a type of stage in the spark that produces data for shuffle operation.

The output of this stage acts as an input for the other following stages.

In the above code, Stage 0 will act as the ShuffleMapStage since it produces data for shuffle operation, which acts as an input for Stage 1.

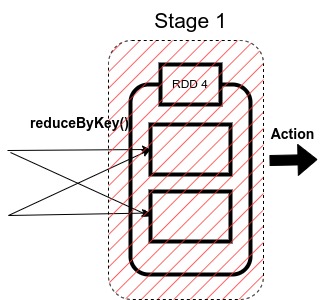


2. ResultStage in Spark

The final stage in a Job executes an action operation by running a function (in our example, the action operation is collect) on an RDD.

It computes the result of the active operation.

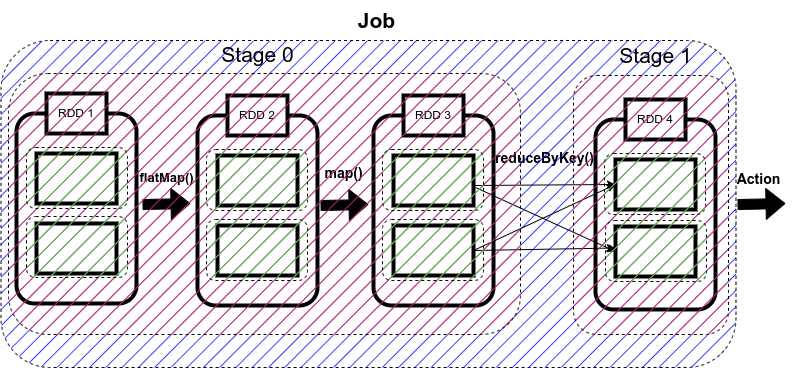
Stage 1 in our example acts as a ResultStage since it gives us a result of an action operation performed on an RDD. In our code, after the data shuffling, similar keys got grouped using reduceByKey() function, so this stage gives us the final result of our code using the collect (action operation) function.

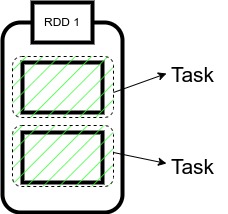


A stage is further a group of tasks executed together. Now we will go through what a task is.

Spark Tasks

The single computation unit performed on a single data partition is called a task. It is computed on a single core of the worker node.





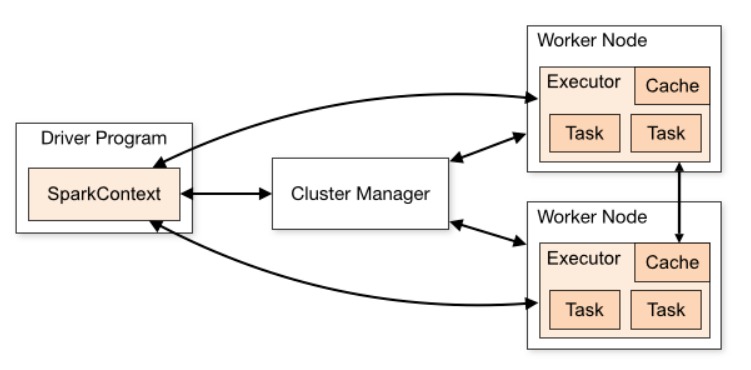
Whenever Spark is performing any computation operation like transformation etc, Spark is executing a task on a partition of data.

Since in our code, we have two partitions of data here, therefore, we have two tasks here.

Each is computing the same operation on a different partition in parallel on a different core of the worker node.

**Some important points to note:**

1. The cluster manager assigns each worker node resources to execute the tasks.



2. A core is the CPU’s computation unit; it controls the total number of concurrent tasks an executor can execute or run.

Suppose if the number of cores is 3, then executors can run 3 tasks at max simultaneously.

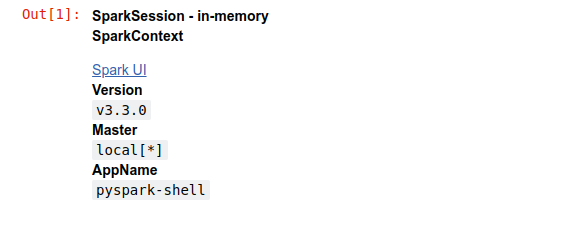
3. Executors are responsible for executing tasks individually. Parallel processing of the tasks by an executor depends upon the number of cores assigned to it, as mentioned in the second point.

4. Each working node has Cache memory for storage, and as soon as a result is computed, it is sent to the driver’s program.

So this is how a Spark application is converted into Job, which is further divided into Stages and Tasks.

Implementation: Jobs, Stages, and Tasks

Now we will be executing the sample code we took as an example to see how Spark’s internal execution works.



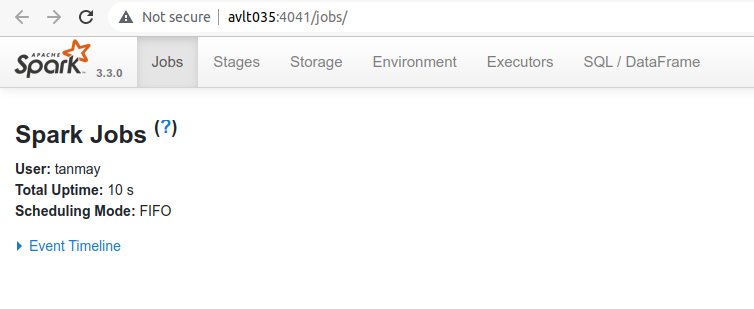
So firstly, we will import the Spark session and then will create the object of this Spark session as ‘spark’ as shown in the above code

When we view this object, it will show the configuration of our Spark clusters like Version, App name, and master.

We can also see Spark UI in this object. It is a graphical user interface that allows us to view all our Spark Application’s jobs, stages, and tasks.

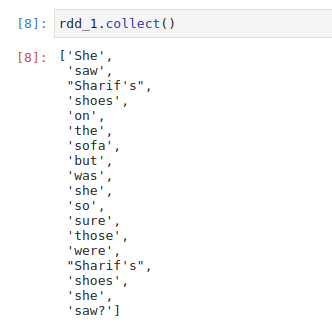
Let’s click on it.

We can see that a new window opens, but in that, there is nothing to view as we haven’t run our application yet.

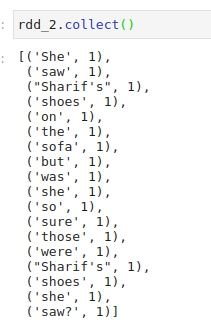


Now we will run our code to output the count of each word in a sentence. We will explicitly create a SparkContext object (it is always necessary to create a SparkContext object to work with RDDs) as we are working with the RDDs, as shown in the code.

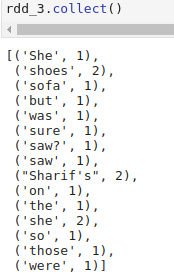
Then we will create an RDD with the name rdd\_0, which will contain our sentence passed in the form of a list followed by our first transformation i.e flatMap(), which will split our sentence into a list of words.



After this, we apply map transformation to create a pair RDD in which we assign value 1 to each word and save it in rdd\_2.



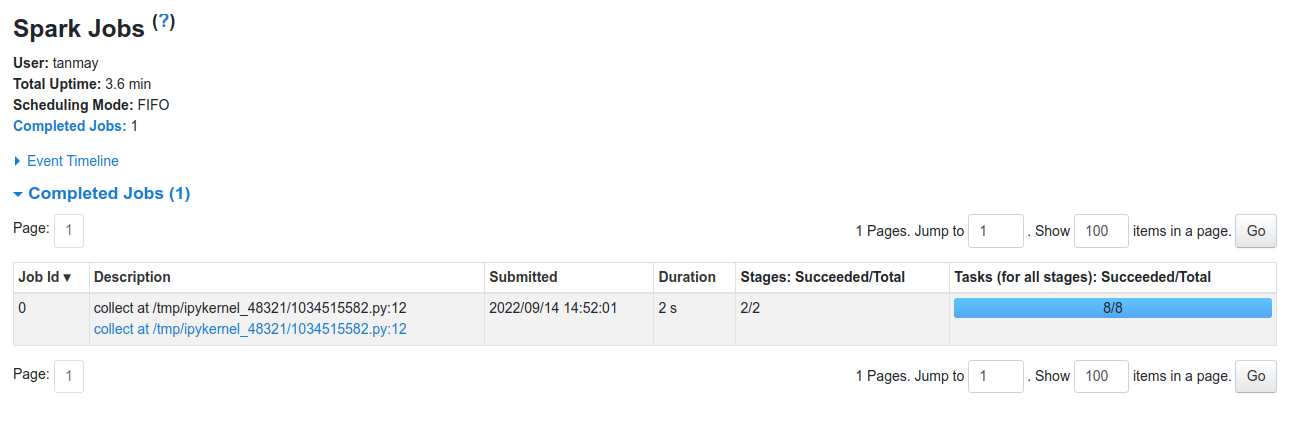
Now to count the occurrence of each word, we will use reduceByKey() method to sum the values of similar words, and we will store it in rdd\_3.



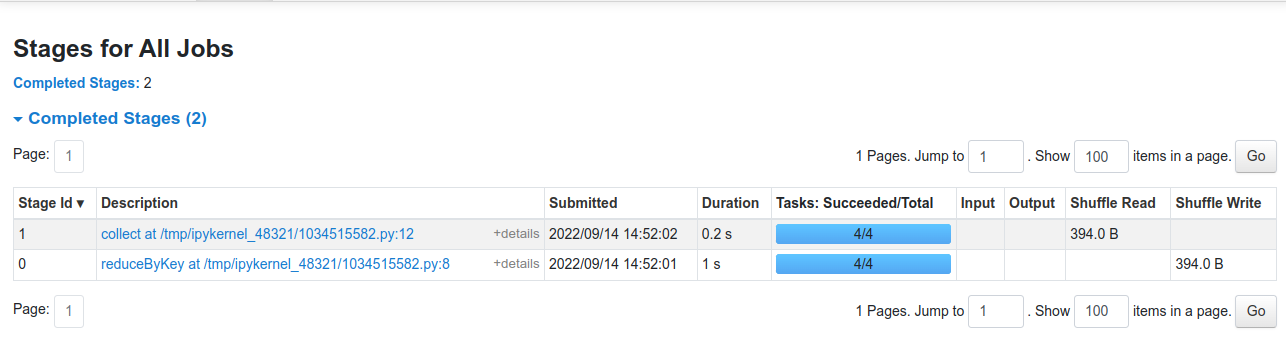
This is how our code is working, now, after executing it when we open the Spark UI, we can see that one job has been completed successfully. It also contains information about Stages, Tasks, Storage, etc, which we can open by clicking on the corresponding tab.

spark

So in our case, we have one job which has been completed successfully.



When we click on the stages tab, we get the information regarding each stage. In our case, there will be two stages. It displays the information regarding the amount of data shuffled. Time is taken to complete a stage, etc.



Similarly, we can see information regarding the tasks that got executed while running the Spark Application.

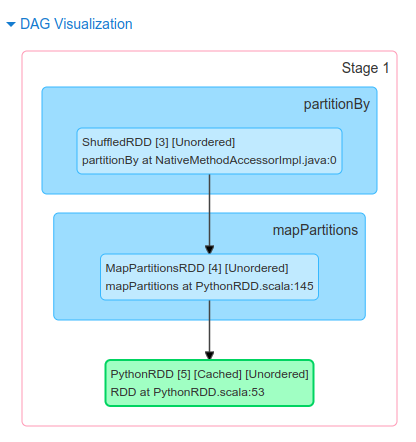
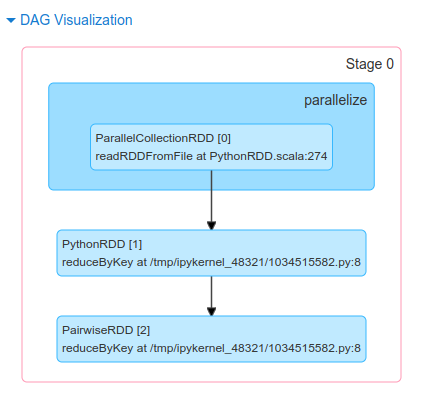
DAG and DAGScheduler

Now the question arises of who is responsible for dividing jobs into multiple stages, optimizing our application, what operations can be performed serially or parallelly, how to execute our application, etc.

The answer to this question is DAG.

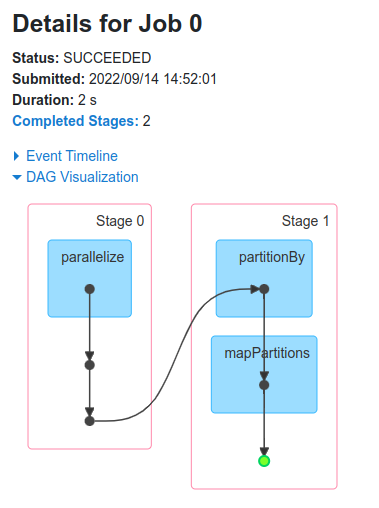
DAG refers to Directed Acyclic Graph whose vertices represent an RDD, and the edges represent the operation on that RDD.

As we write our Spark Application, Spark converts this into a DAG that is executed by the core engine.



As soon as an action function is called, the created DAG is submitted to the DAG scheduler, which is a high scheduling layer that implements stage-oriented scheduling.

It completes the computation and execution of stages for a job. It also keeps track of the RDDs and runs jobs in minimum time.



It also pipelines the tasks into stages. After this, the DAG Scheduler sends the tasks to Task Scheduler, which further sends them to the Cluster manager.

The cluster manager provides resources to the worker node executors required to run the tasks.

Conclusion

I hope through this article; you were able to get an understanding of how a Spark Application is executed. It aims at creating a basic understanding of Jobs, Stages, and Tasks, followed by a brief introduction to DAG and DAGScheduler.