**Documentation of 20th May 2021**

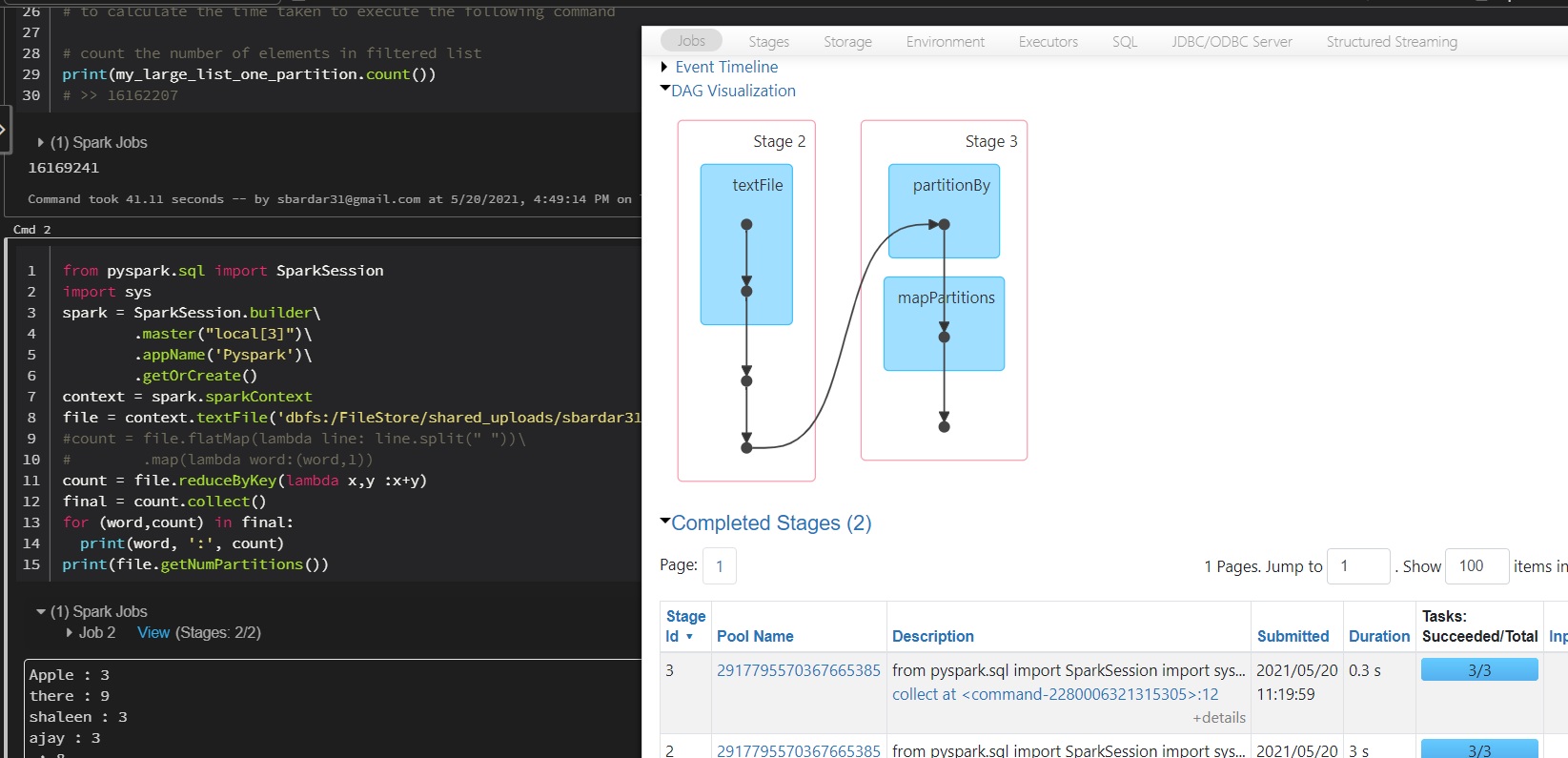
**How RDD is Partitioned internally ->**

First of all, RDD is partitioned through spark, manually we can tell spark how many partitions we want to have.

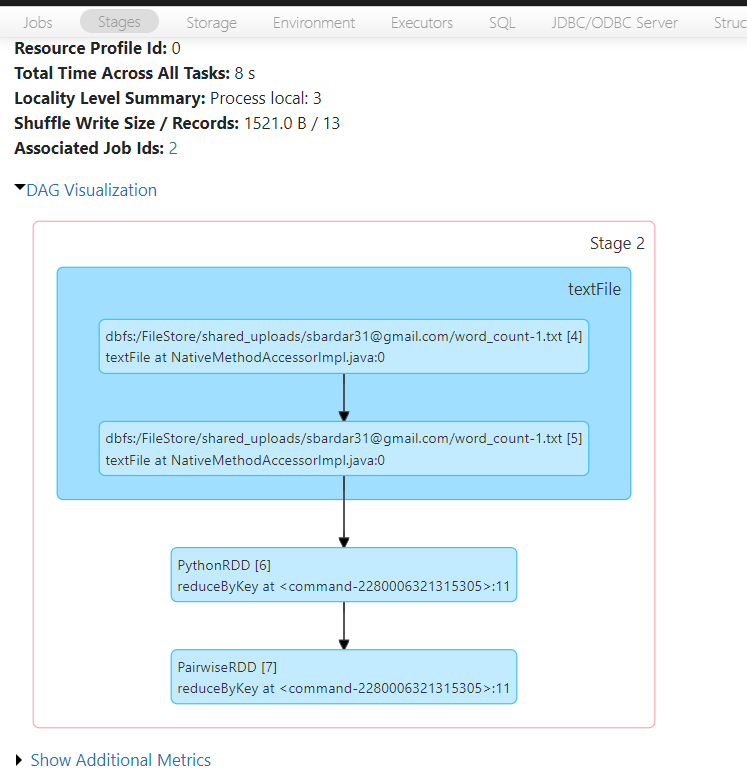
Partitions are very important in Spark as you can compute parallelly efficiently if your dataset or RDD is partitioned into nodes along a cluster. It is the best to have the no of partitions = no of executers which Is provided by Spark by default.

**For example** – If a have a list of integers ranging from (0-20000) and I want to filter from those integers which are greater than 200, and if we have 10 executers than the spark will divide the RDD or dataset into 10 partitions along nodes so that every executer can have one partition to work upon no executer will be on hold and no partition will be left unexecuted.

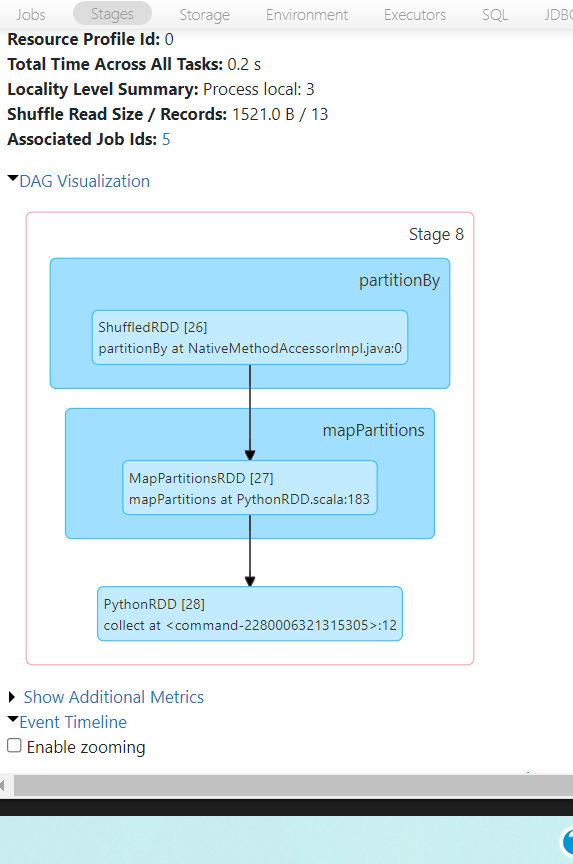
Snippets from word count describing RDD transformations ->

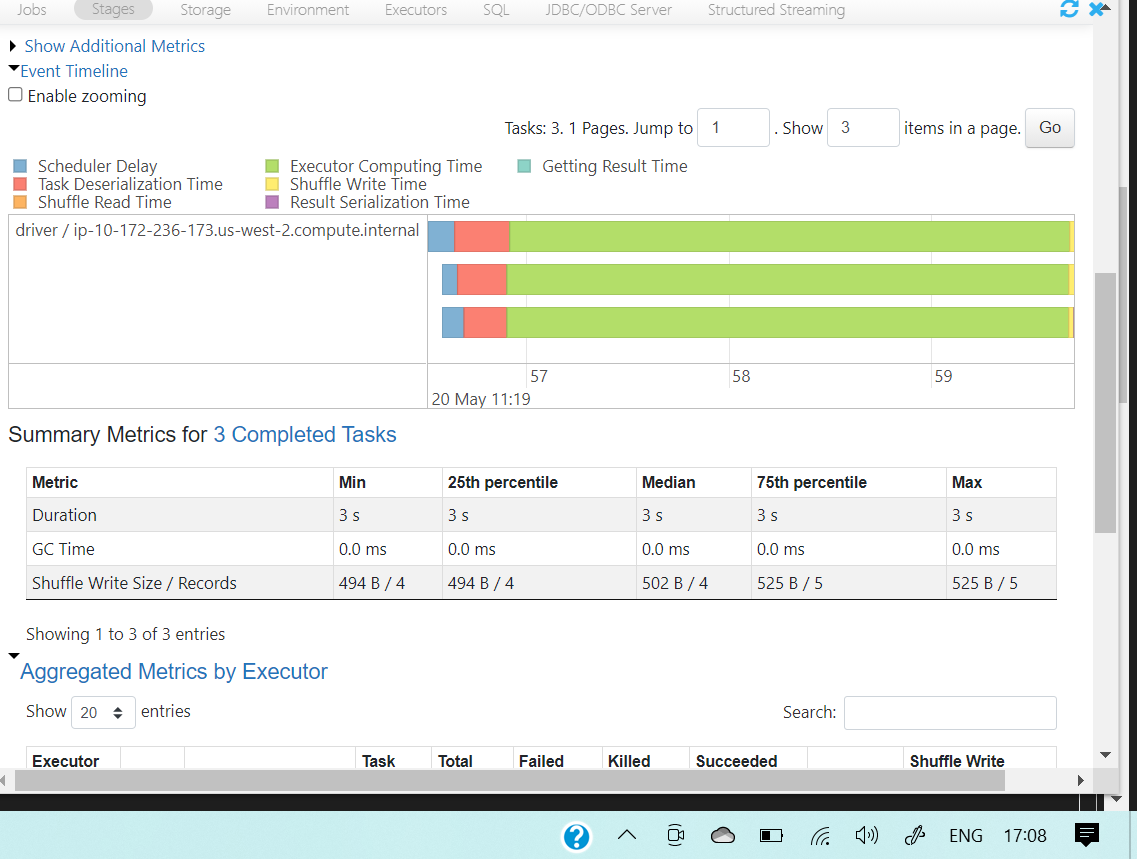


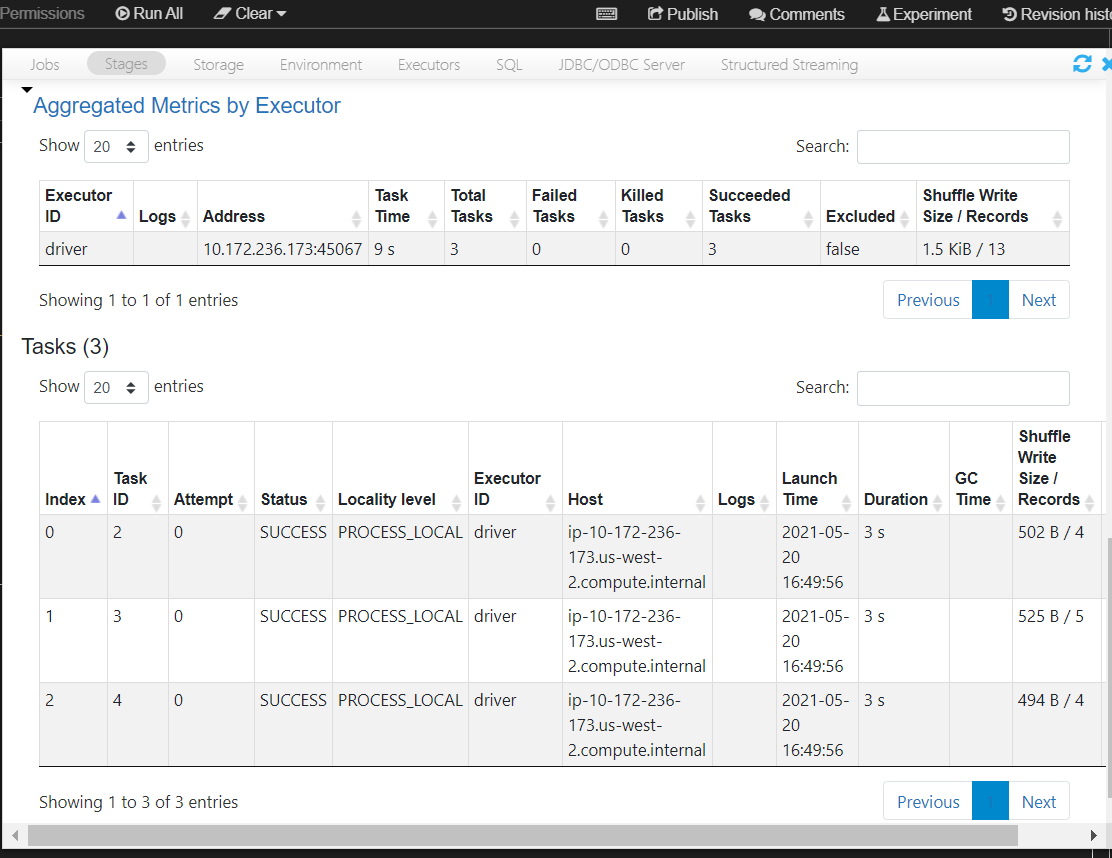
Shows the Dag for RDD transformation – The Black dots represent the Transformation occurs on RDD



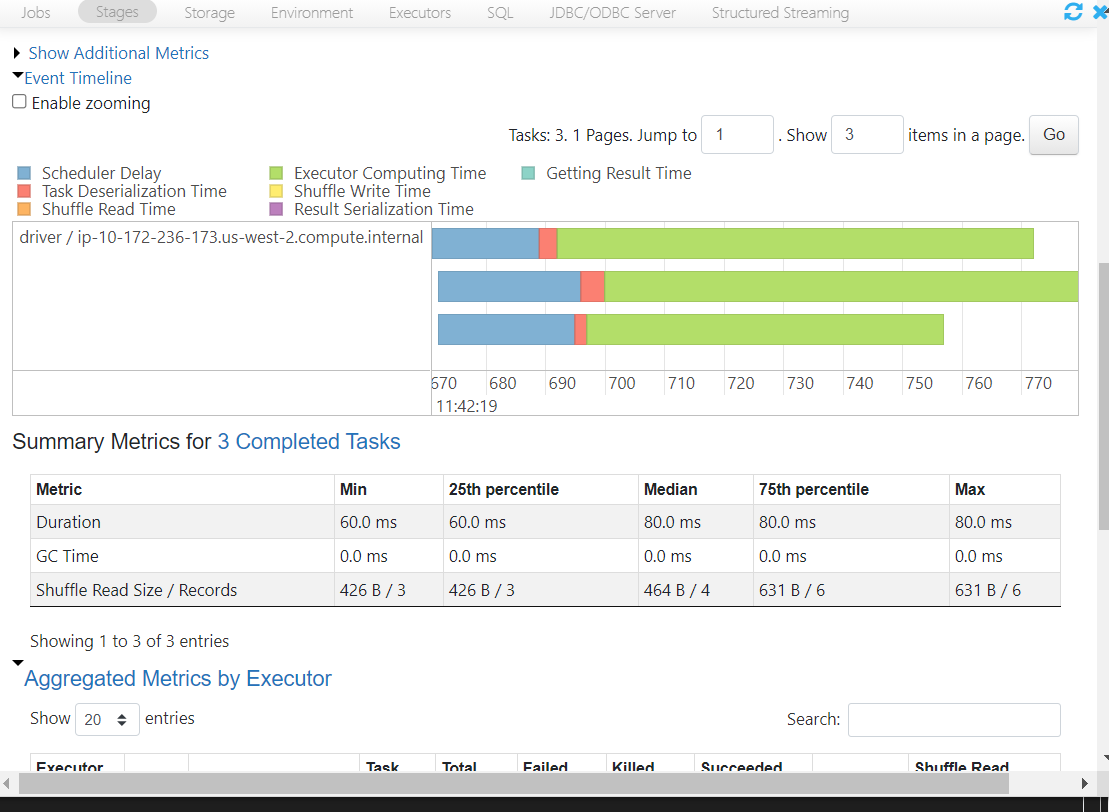
You can See here PythonRDD is transformed to PairwiseRDD



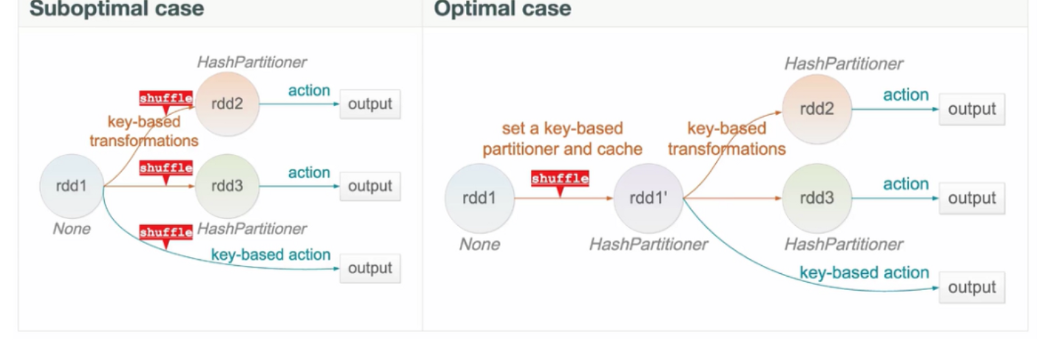




This Shows the Executer execution time and details – Here only Driver is used.



Why partitioned Data ?



*Partition helps in localizing the data and reduce the data shuffling across the network nodes reducing network latency which is a major component of the transformation operation thereby reducing the time of completion.*

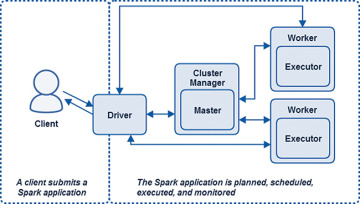
*1) Partition done every time transformation is done*

*2) Partition done before hand.*

RDD partitioning has 2 methods -

1. Hash Partitioning – Based on key value
2. It partitions data either based on some sorted order OR set of sorted ranges of keys, tuples with the same range will be on the same machine.

**How Executer execute a partition in a cluster and related to JVM ->**



This is the Workflow of client/server architecture of Spark.

* Driver- Distributes the work along workers and manages the cluster. One of the main functions of the Driver is to plan the application. The Driver takes the application processing input and plans the execution of the program. The Driver takes all the requested transformations (data manipulation operations) and actions (requests for output or prompts to execute programs) and **creates a directed acyclic graph (DAG)** of nodes, each representing a transformational or computational step. **Driver is a Java process. This is the process where the main() method of our Scala, Java, Python program runs. It executes the user code and creates a SparkSession or SparkContext and the SparkSession is responsible to create DataFrame, DataSet, RDD, execute SQL, perform Transformation & Action, etc.**
* Cluster manager has the partitions that executer along with the driver execute. The Master and the Cluster Manager are the central processes that monitor, reserve, and allocate the distributed cluster resources (or containers, in the case of YARN or Mesos) on which the Executors run.
* Task - A task is the smallest unit of schedulable work in a Spark program.
* Stage - is a set of tasks that can be run together. Stages are dependent upon one another; in other words, there are stage dependencies.

Executer – Executer is place where spark DAG tasks run. Executers reserves the memory resources and computational resources on worker nodes (slave nodes). Just like a thread in **JVM the executer gets terminated as soon as the application ended and associated only with the application.**  As Spark is built on Scala and Java, so the JVM multithreading is related to Spark executer in a way that they both have the same characteristics. Just like the JVM main() function one thread activates automatically the Executer – Driver will come in effect in Spark.

Working of Spark real time – That will clear more of the doubts and built concepts on how things work here ->

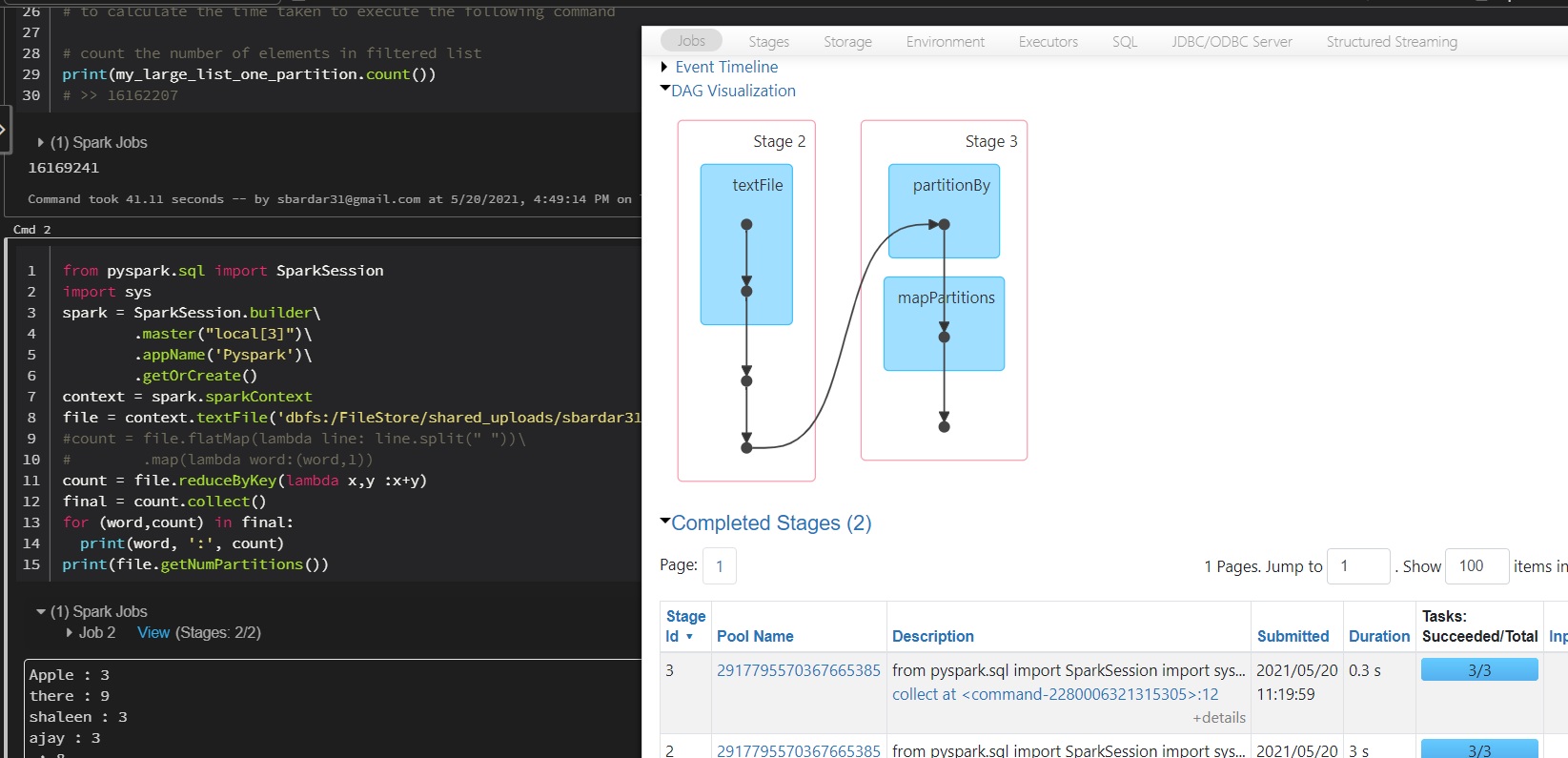
1. Let’s say a user submits a job using “spark-submit”.
2. “spark-submit” will in-turn launch the Driver which will execute the main() method of our code.
3. Driver contacts the cluster manager and requests for resources to launch the Executors.
4. The cluster manager launches the Executors on behalf of the Driver.
5. Once the Executors are launched, they establish a direct connection with the Driver.
6. The driver determines the total number of Tasks by checking the Lineage.
7. The driver creates the Logical and Physical Plan.
8. Once the Physical Plan is generated, Spark allocates the Tasks to the Executors.
9. Task runs on Executor and each Task upon completion returns the result to the Driver.
10. Finally, when all Task is completed, the main() method running in the Driver exits, i.e. main() method invokes sparkContext.stop().
11. Finally, Spark releases all the resources from the Cluster Manager.

So that is how the JVM is related to Spark Executer.

**How the Lambda function is applying to each micro partition on executer ->**

* The RDD is made as soon as there is a transformation or an action called by the driver.
* Then driver with the help of cluster manager and executor partitioned the RDD, gets allocated into nodes.
* Now as soon as the lambda function (which is used to compute data line by line) will come into action – the partition of RDD which is residing in the executor gets transformed into another RDD into the same partition using the Lineage.
* Now when all the execution was done by every worker node in the executor, the partition which is newly transformed (residing into the memory allocated by worker nodes) gets mapped with one another and give the result to the driver.

This is done with the help of DAG which created a logical graph of how the data should be flowed with minimum of fuss.

Snapshots of DAG - 

Represent two DAG – 2 Stages – Dot denotes the tranformation of RDD.

