What is Spark?

Spark Programming is nothing but a general-purpose & lightning fast cluster computing platform. It is an open source, data processing engine. Ability to efficiently execute streaming, Machine learning and SQL workloads which requires fast-iterative access to datasets. Spark can perform batch processing and stream processing. It is designed for data science and its abstraction and makes data science easier. It can cache data set in memory and speed up iterative data processing.

Spark is independent of Hadoop since it has its own **cluster management** system. Basically, it uses Hadoop for storage purpose only. Spark is written in Scala still offers rich APIs in Scala, Java, Python, as well as R.

comparing Spark with Hadoop, it is 100 times faster than Hadoop In-Memory mode and 10 times faster than Hadoop On-Disk mode.

Why Spark?

- To perform batch processing, we were using **Hadoop MapReduce**
- Also, to perform stream processing, we were using Apache Storm / S4.
- Moreover, for interactive processing, we were using Apache Impala / Apache Tez.
- To perform graph processing, we were using Neo4j / Apache Giraph.

Hadoop MapReduce Limitations

- It's based on disk based computing.
- Suitable for single pass computations -not iterative computations.
- Needs a sequence of MR jobs to run iterative tasks.
- Needs integration with several other frameworks/tools to solve big data use cases.
- Need Apache Storm for stream data processing.
- Need Apache Mahout for machine learning.

Performance:

- SPARK PROCESSES DATA IN MEMORY(RAM), WHILE MR PERSISTS BACK TO DISK, AFTER A MAPREDUCE JOB.

- SO SPARK SHOULD OUTPERFORM MR.
- NONETHELESS, SPARK NEEDS A LOT OF MEMORY.
- IF DATA IS TOO BIG TO FIT IN MEMORY, THEN THERE WILL BE MAJOR PERFORMANCE DEGRADATION FOR SPARK.

- MR KILLS ITS JOB, AS SOON AS IT'S DONE.
- SO IT CAN RUN EASILY ALONGSIDE OTHER SERVICES WITH MINOR PERFORMANCE DIFFERENCES.

- STILL SPARK HAS AN ADVANTAGE, AS LONG AS WE ARE TALKING ABOUT ITERATIVE OPERATIONS ON THE DATA.

What is RDD?

It is a primary abstraction in spark and it is the core of spark.

One could compare RDDs to collection in programming(like in python we have list, Dictionaries, tuple but if we talk about spark we have RDDs, DF, DS).

RDD is immutable means if you create one RDD you can not change. Partitioned collection of records means if you have 10gb of RDD then it can be partitioned into smaller ones(like 10gb, 10gb,……..).

It can only be created by reading data from stable storage like HDFS or by transformation on existing RDDs.

RDD is an acronym for Resilient Distributed Dataset. It is the fundamental unit of data in Spark. Basically, it is a distributed collection of elements across cluster nodes. Also performs parallel operations. Spark RDDs are immutable in nature. Although, it can generate new RDD by transforming existing Spark RDD.

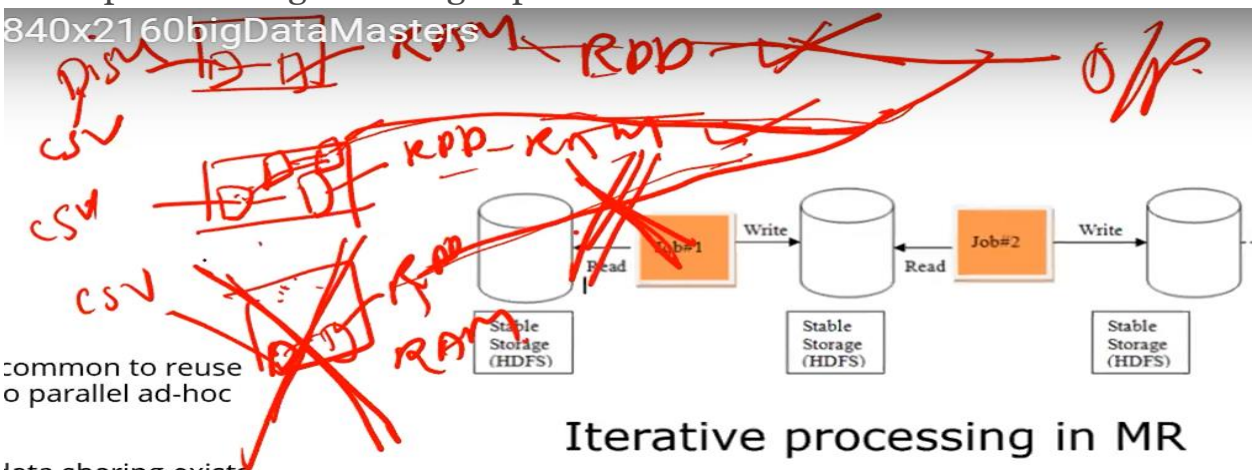
Ways to create Spark RDD

- **Parallelized collections:** By invoking parallelize method in the driver program.

- **External datasets:** One can create Spark RDDs, by calling a textFile method. Hence, this method takes URL of the file and reads it as a collection of lines. In this, if we are reading file from the disk it is in the disk. OR Text file is in the disk.
- **Existing RDDs**: we can create new RDD in spark, by applying transformation operation on existing RDDs.

Features of RDD

- Resilient, fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures.



  Eg. suppose we run a query, and there are three Data Nodes/Node Manager and each data node taking existing data from CSV file and make new RDD due to some issue the third node will gone. It will not fail the entire job ... it just search for replicas and make it new node and make complete the processing. (HDFS having replicas same concept will apply here in spark)
- For iterative distributed computing, it's common to reuse and share data among multiple jobs or do parallel ad-hoc queries over a shared dataset.
- The persistent issue with data reuse or data sharing exists in distributed computing system(like MR) -that is, you need to store data in some intermediate stable distributed store such as HDFS or Amazon S3.
- This makes overall computation of jobs slower as it involves multiple IO operations, replications and serializations in the process.
- Distributed with data residing on multiple nodes in a cluster.
- Dataset is a collection of partitioned data with primitive values or values of values, e.g tuples or other objects.

Traits of RDD

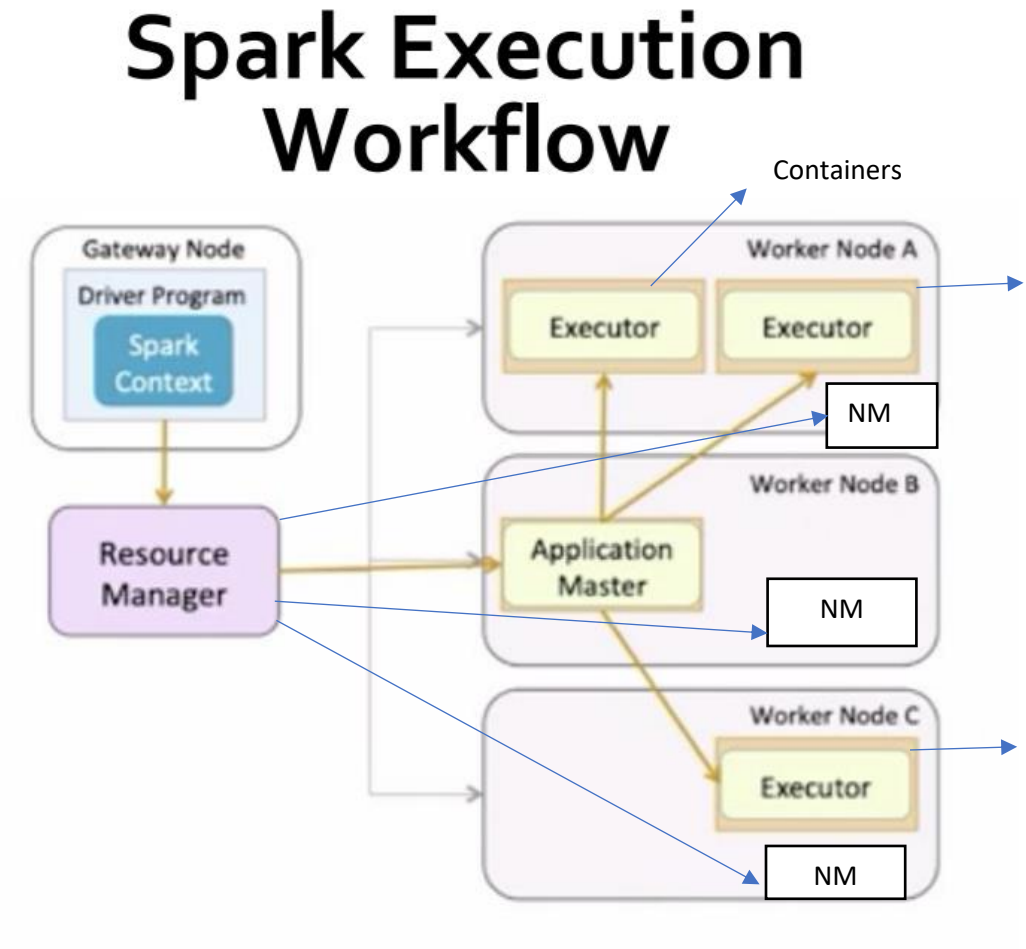
- In-Memory, data inside RDD is stored in memory as much (size) and long (time) as possible.
- Immutable or Read-Only, it does not change once created and can only be transformed using transformations to new RDDs.
- **Lazy evaluated**, the data inside RDD is not available or transformed until an **action** is executed that triggers the execution.
- Cacheable, you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed).
- Parallel, process data in parallel.
- Typed-RDD records have types, Long in RDD[Long] or (Int, String) in RDD[(Int, String)].
- Partitioned-records are partitioned (split into logical partitions) and distributed across nodes in a cluster.
- Location-Stickiness-RDD can define placement preferences to compute partitions (as close to the records as possible).
- RDD Supports Two Kinds of Operations
- **Actions** -operations that trigger computation and return values
- **Transformations** -lazy operations that return another RDD.

A **Transformation** is a function that produces new **RDD** from the existing DDs but when we want to work with the actual dataset, at that point **Action** is performed. When the action is triggered after the result, new RDD is not formed like transformation.

Spark Core Concept

- Spark is built around the concepts of Resilient distributed Datasets and directed acyclic graph representing transformations and dependencies between them.
- Spark Application(often referred to as Driver Program or Application Master/YARN)at high level consists of spark context and user code which interacts with it creating RDDs and performing series of transformation to achieve final results.

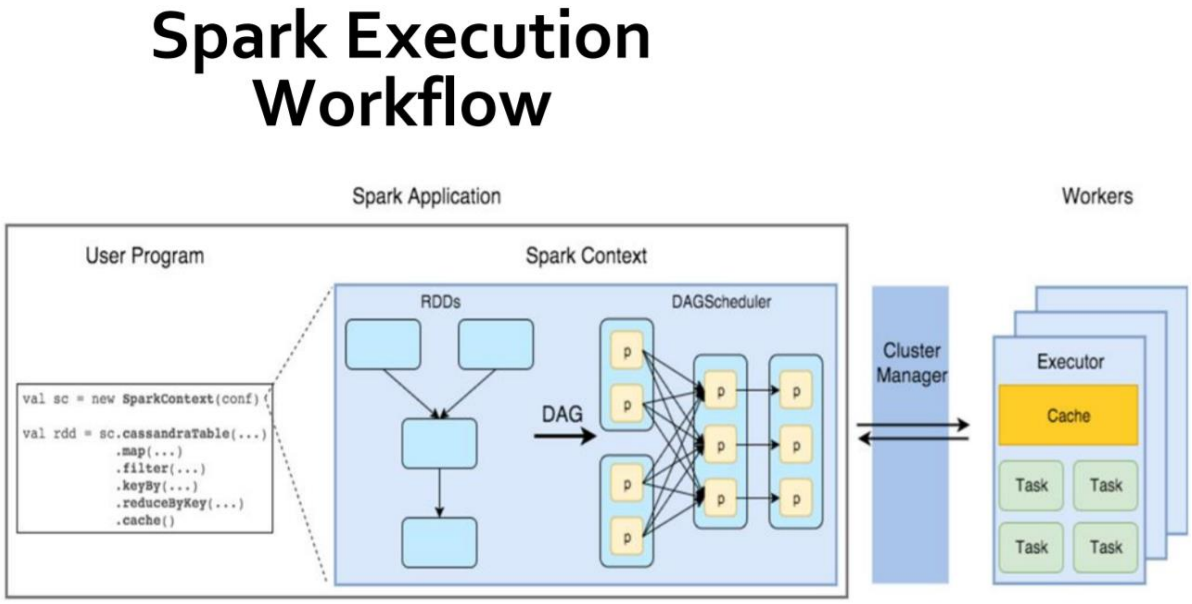
- These transformations of RDDs are then translated into DAG and submitted to Scheduler to be executed on set of worker nodes.



In the above diagram, Spark context(sc) tells the compiler that we are not using python anymore means we are using pySpark onwards. Driver Program is the base of program it is controlling the execution of job. It is entry point. Application Master is created it said that I need executors or workers to do task. Executor is a program/process which does the processing of the spark application.
Ques: Why we have two executor node in the same data node/worker node? Why we don't have single and big executor node?
Ans: If we reduce the number of executor or if you use same number of executor equal to the same number of worker the parallelism will be compromised (Eg if we have 100Gb of data and we have 5nos of executor and that 100Gb of data is divided into 5 chunks then and only parallelism is maintained but if we only have one executor then all work load is on one

worker node and it will take lot of time). Keep in mind that we can not increase too much number of executor because if more number of executor more Input/output overhead is there. Even we use ram then it will take a lot of time. Balanced should be maintained between executor and worker node.

Spark Execution Workflow



Here we are using spark context which we know that this is the entry point to spark. In the spark context section, we describe RDD and its DAG. DAG shows that how many partition created of RDD1 and RDD2 and so on (eg. Rdd1.parallelize(data,2)). And then we join RDD1 and RDD2 with some following transformation RDD3 got created. RDD3 give me shuffle because need to be join. After that RDD4 will created. Cluster manager or YARN in this each partition will run its separate task. Cache is saving the intermediate state of RDD so that if I need RDD1 while working with RDD2 cache/presist will save the state.

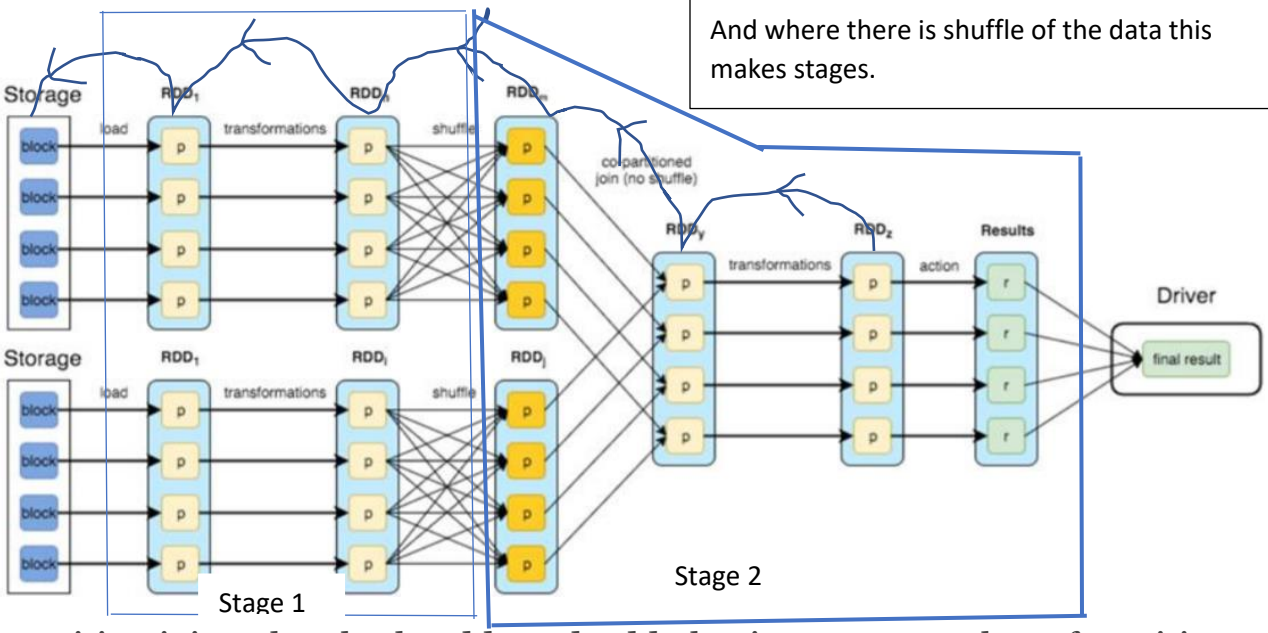
- User code containing RDD transformations forms Direct Acyclic Graph
- DAG then split into stages of tasks by DAG Scheduler.
- Stages combine tasks which don't require shuffling/repartitioning.
- Tasks run on workers and results then return to client.

Let Us Analyze a DAG

Let Us Analyze a DAG

These are the following dependencies that one stage is dependent on the previous stage. Dependencies needs shuffle of the data.

And where there is shuffle of the data this makes stages.

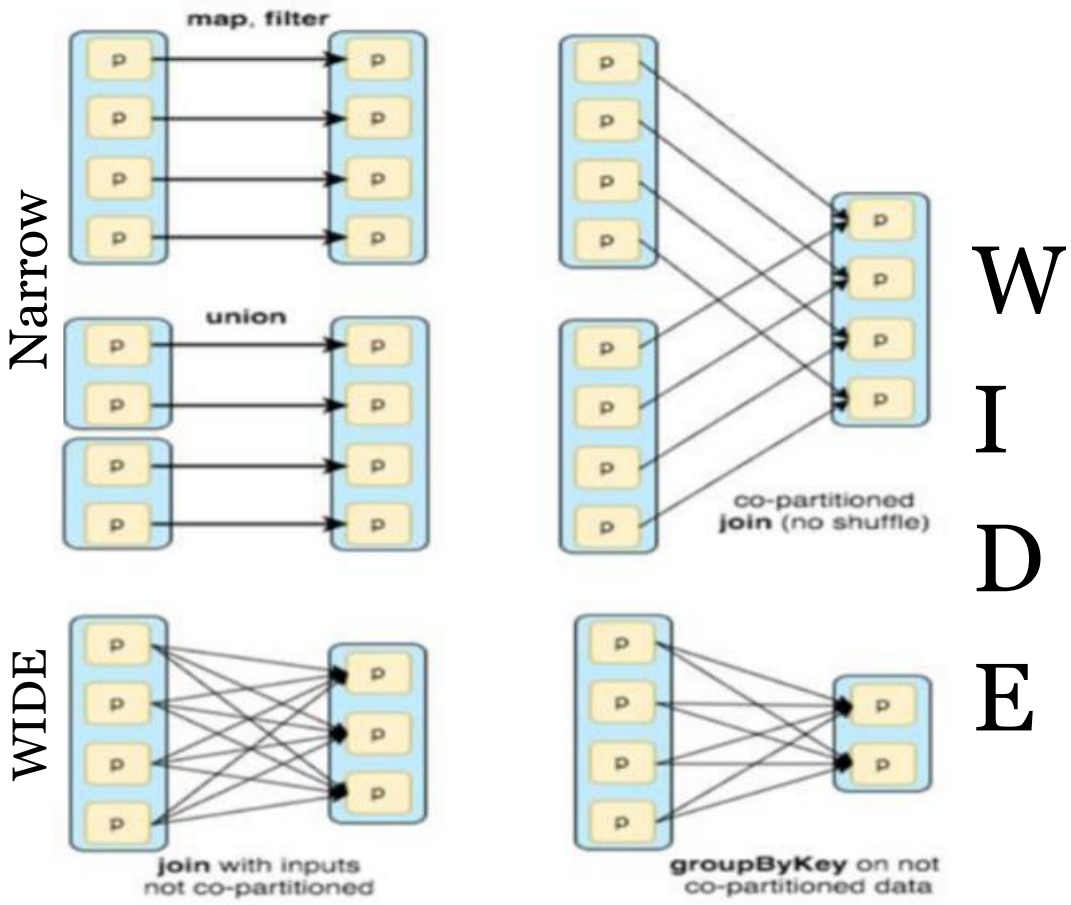


Co-partition join: when both Rdd1 and Rdd2 having same number of partitions. To make the same number of partitions we need to do shuffle the partition.

- Any data processing workflow could be defined as reading the data source.
- Applying set of transformations and materializing the result in different ways.
- Transformations create dependencies between RDDs.
- The dependencies are usually classified as "narrow" and "wide".

Narrow (pipelineable)

- each partition of the parent RDD is used by at most one partition ofthe child RDD.
- Allow for pipelined execution on one cluster node.
- Failure recovery is more efficient as only lost parent partitions need to be recomputed

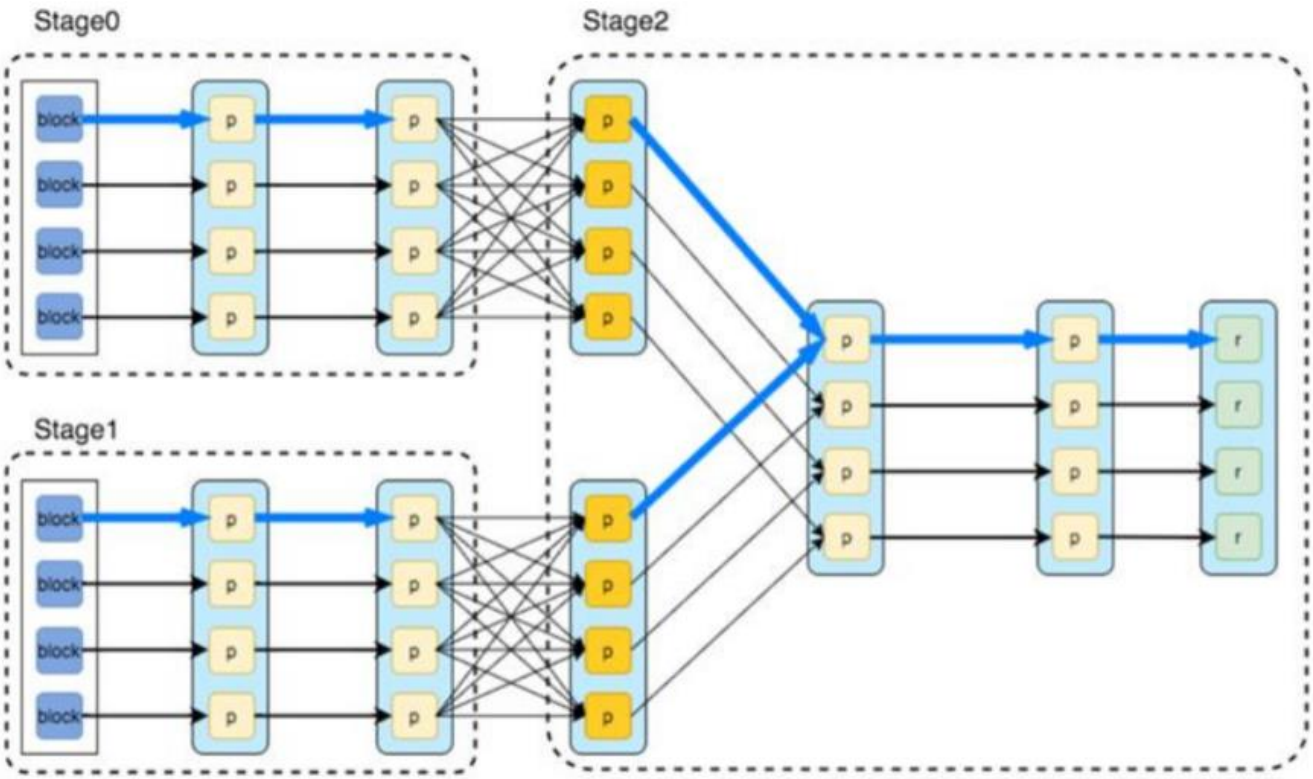


Wide (shuffle): It is little bit slower than narrow because of shuffling in wide transformation.

- multiple child partitions may depend on one parent partition and vise-versa
- Require data from all parent partitions to be available and to be shuffled across the nodes.
- If some partition is lost from all the ancestors a complete re-computation needed.

Splitting DAG Into Stages

Spark stages are created by breaking the RDD graph at shuffle boundaries. OR Where we have wide transformation/Shuffle we have the different-2 stages.



LIST OF NARROW VS WIDE TRANSFORMATION

Transformations with (usually) Narrow dependencies:

- map
- mapValues
- flatMap
- filter
- mapPartitions
- mapPartitionsWithIndex

Transformations with (usually) Wide dependencies: (might cause a shuffle)

- cogroup
- groupWith

•join

•leftOuterJoin

•rightOuterJoin

•groupByKey

•reduceByKey

•combineByKey

•distinct

•intersection

•repartition

•coalesce

RDD Action

when we want to work with the actual dataset, at that point action is performed. When the action is triggered after the result, new RDD is not formed like transformation. They trigger execution of RDD transformation to return values. Thus, Actions are Spark RDD operations that give non-RDD values. The values of action are stored to drivers or to the external storage system. It brings laziness of RDD into motion. This action evaluates the RDD lineage graph.

An action is one of the ways of sending data from *Executer* to the *driver*. Executors are agents that are responsible for executing a task. While the driver is a JVM process that coordinates workers and execution of the task.

Link:click here to see all actions and transformations

RDD Transformation

Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. Thus, the so input RDDs, cannot be changed since RDD are immutable in nature. **Transformations are lazy** in nature i.e., they get execute when we call an action. They are not executed immediately. Two most basic type of transformations is a map(), filter().