RDDs are the basic building blocks of spark.

In Spark, data is represented as RDD.

RDD stands for Resilient Distributed Dataset

Step 1 : Specify the number of executors you want. Then the file will be split and given to the executors. The file is in the disk.

Step 2 : Create an RDD by reading the file. Each node(executor) will have an RDD. We can give the location of the file for reading.

Step 3: All the data which is residing in the disk will be copied to the memory.

In hadoop in 99.9% of the situations we cant specify the size of the containers or the number of containers. But in Spark we can do that.

Suppose our file is divided into 4 blocks and kept in 4 machines. If we tell spark that we need 4 executors for processing. There is no guarantee that the executors will be created in these 4 machines itself where data is available. Different machines might be allotted and for processing the data has to be copied to the RAM of the executor. This delays processing initially.

Spark dynamic execution allocation - enable/disable

If 8 systems are launched and only one system is being used.

Enable - It will launch 8 and kill 7

Disable - It will keep all 8

An executor is a JVM

Data in RAM is called RDD.

If there are 4 nodes(executors) it is called as a 4 partition RDD. There can be more than one partition in the executor. Suppose the RAM in the executor is 10GB and the partition size is 1 GB. We can have 10 partitions in a single executor.

The number of partitions depends on the number of cores. Each partition will be using one core. So for a dual core machine there can be maximum 2 partitions.

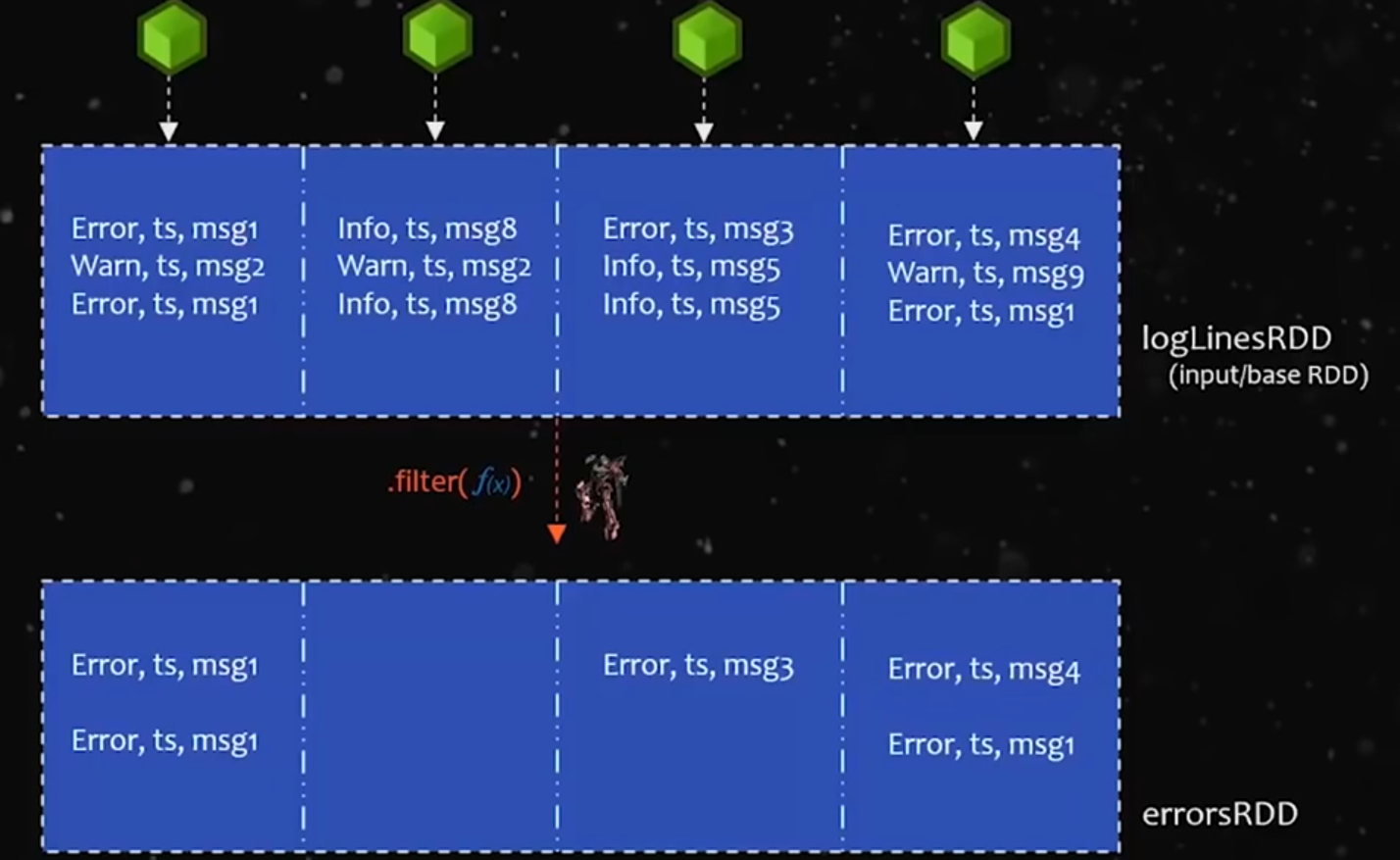
Suppose we have 10GB of RAM. All of it won’t be allocated for RDD.

10% of RAM will be allocated for system calls. It is a container so it has to accept system calls and respond. In the remaining 90% we will be getting only 60%. It is a JVM it has garbage collection, JVM management. Container has to communicate with OS so for that also we require some memory.

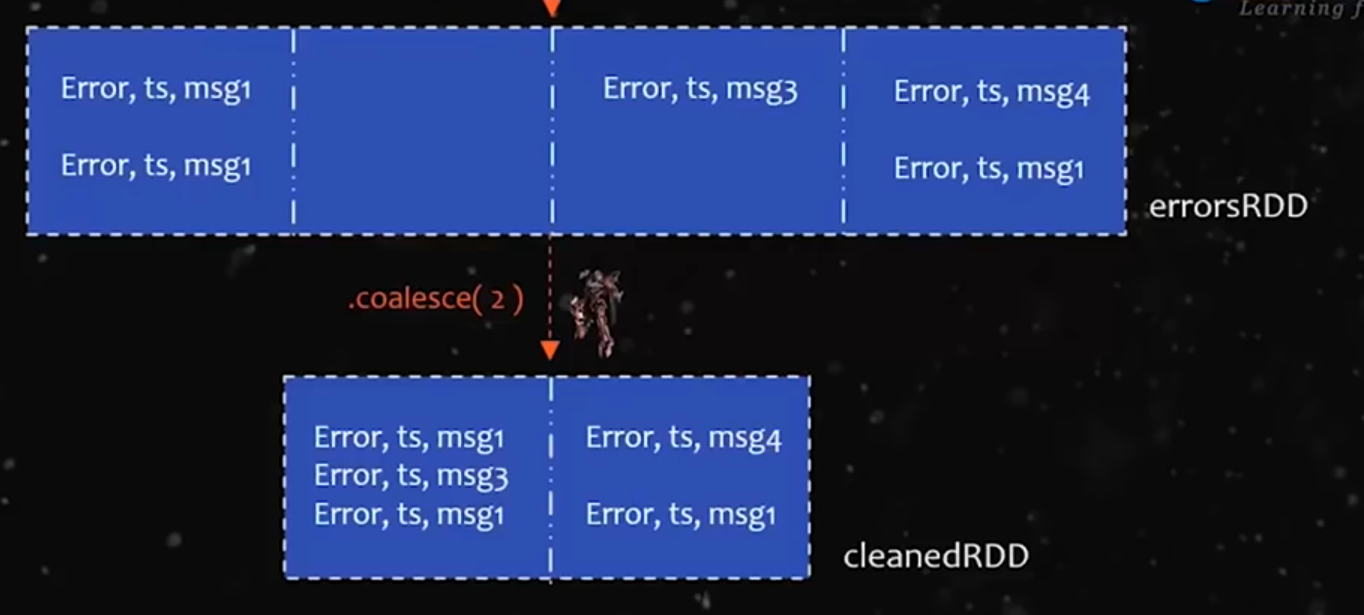
So in reality if we ask for a 10GB container only 5.8GB will be allocated for RDD.

RDDs are immutable

Initially there is a RDD and after we apply a filter function to it we get a new RDD. If we enable dynamic memory allocation in yarn. The old RDD will be deleted.

Filtering only error messages using filter function 

As we can see from the above picture there is one empty executor. So in order to solve this we use a transformation called coalesce. We can specify how many executors we want and the resultant RDD will have that many executors.



Another transformation is repartition which can also be used to increase the number of JVM(executors). We have to provide the number of executors we need.

Repartition can also be used to decrease the number of partitions.

In coalesce there is very little movement of data. Sparsely populated data would simply be moved to other JVMs. But in the case of repartition there will be a complete reshuffle. This is a costly operation because we have to read all the data into the RAM first and then reshuffle.

Coalesce is smarter in this aspect.

Spark follows Lazy Execution.

Spark will not start execution unless we call action. Collect() is the most common action in spark.

Collect() - used to see the final output.

In initial stages we perform filter, coalesce, transform but no process is done.

When we call collect() it will go through the previous RDD(cleaned RDD) and for getting cleanedRDD it has to have errorsRDD. In order to have errorsRDD it needs to have logLinesRDD. For this we have to first read the data from the disk. So now only spark starts to read the data. Because of this behaviour spark follows lazy execution.

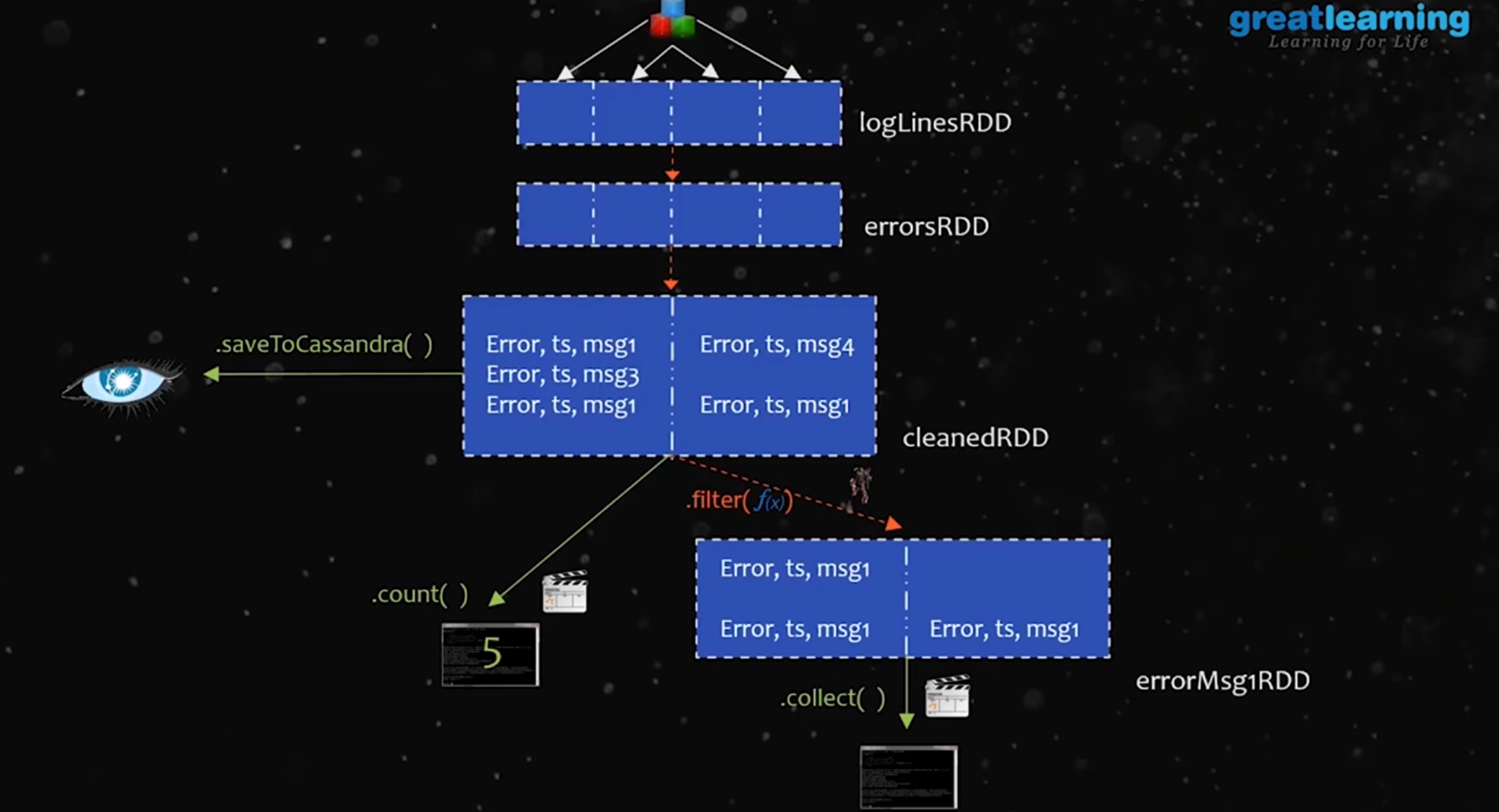


No data will be saved. It will jus display the result on the screen. If u want to do a different process u have to do it again from the first step.

We can use save function to calculate the size of the file.

This will tell us how many containers should be allocated and how much processors are needed.

Spark follows DAG(Directed Acyclic Graph). Spark creates all the steps to be executed in a graph format.



For performing these 3 actions(count(), saveToCassandra() , filter()) we have to run the pipeline 3 times one for each action. But if we see all these 3 actions originate from the cleanedRDD. It doesn’t make sense to run the whole pipeline again and again for every new action. So in order to solve this issue we cache the RDD. We can cache in RAM as well as in HardDisk. Cache will hold the data in memory.

Caching will be in place till the spark program executes. After it completes execution, caching is deleted.

Spark Context Object - Represents the program and when the program completes the object is killed.

unpersist() → its an action which will delete the cache.

Disadvantage of RDD:

1. RDD doesn’t have a strict schema. It doesn’t perform well against structured data. It can handle structured data but spark can’t optimise the code.

For that we use Spark SQL or dataframes. Spark SQL is more powerful than core spark. It can perform optimisations to the code.

Yarn has no role to play in local mode. Everything is allocated by OS in local mode.

sc.\_conf.getAll() - to get all information about the jobs, environment from CLI

Spark Optimisation Engine

Each query we run in spark sql is going to be passed through an optimisation engine that is designed to give the best possible performance. It can adjust the order of the business logic or it can skip unnecessary data and it can also find the most cost efficient way to deliver the results.

