<https://www.youtube.com/watch?v=xS82xt6wvzw>

Spark

Apache **Spark** is an in-memory distributed data processing engine and **YARN** is a cluster management technology. ... As Apache **Spark** is an in-memory distributed data processing engine, application performance is heavily dependent on resources such as executors, cores, and memory allocated

Spark is nothing but distributed computing framework. To leverage the framework we need to learn API categorized into different modules and build applications using supported programming languages(like Scala,Python,Java etc)

Spark Modules

Core – Transformations and actions

Spark SQL and Data Frames

Structured Streaming

Machine learning Pipelines

GraphX Pipelines

Spark Data Structures

Resilient Distributed datasets (an in memory distributed collection)

Data Frame (a wrapper on top of RDD with structure)

RDD

Resilient Distributed Datasets (**RDD**) is a fundamental data structure of **Spark**. It is an immutable distributed collection of objects. Each dataset in **RDD** is divided into logical partitions, which may be computed on different nodes of the cluster. ... Formally, an **RDD** is a read-only, partitioned collection of records.

Syntax :

Rdd = sc.textFile(“hdfs path”)

All the hdfs blocks considered as rdd partitions.

It will just read the data in streaming fashion and then output will be saved into memory as available if not it is saved into someware else

* Data from files will be divided into RDD partitions and each partition processed by separate task
* By default it will use HDFS block size (128 MB) to determine partition
* We can increase(cannot decrease) number of partitions by using additional parameter in sc.textFile
* By default when data is loaded into memory each record will be serialized into java object
* We can persist the RDD partitions at different storage levels
* Memory\_ONLY(default)
* MEMORY\_AND\_DISK
* DISK\_ONLY

\*\*Rdd is nothing but serialized java objects of input data

\*\*Rdd partition is nothing but serialized java objects for a given block

\*\* block is a part of text data

\*\*RDD is nothing but bunch of partitions and each rdd partition hava serialized java object which are created by reading data from file into memory

And then once the first rdd is created once it is process they will be rotating as the data is being processed and finally again those java objectswill be deserialized and will be return into file.

1 gb of text data is divided into eight 128mb blocks AND one 28mb block

Important concepts of Rdd:-

RDD persistence:

>>> lines = sc.textFile("/public/randomtextwriter/part-m-00000")

>>> from pyspark import StorageLevel

>>> lines.persist(StorageLevel.MEMORY\_ONLY)

Once the data is persisted reprocess the data again and again

Once the sparkContext is close what ever data persisted is gone

. **DataFrame**

A **DataFrame** is a distributed collection of data organized into named columns. It is conceptually equal to a table in a relational database. Data frames is nothing but collections with structures

DataFrame nothing but a RDD with structure. internally everything is Rdd with respective spark but DataFrame is wrapper on top of typical Rdd where a structure is on defined on it

DF = spark.read.csv(“Hdfs path”)

* Flexible APIs(Data frame native operations as well as SQL)
* Code will be readable
* Better organized and manageable
* Uses latest optimizers
* Process data in binary format
* Can generate execution plans on statistics collected

Spark Framework And Execution Modes

* Driver Program
* Spark Context
* Executors
* Executor Cache
* Executor Tasks
* Job
* Stage
* Task(Executor Tasks)

Diff executor mode supported by spark are

1. Local(for development)
2. Standalone(for development)
3. Mesos (production)
4. YARN (production)

spark work flow

* Using spark-submit, the user submits an application.
* In spark-submit, we invoke the main() method that the user specifies. It also launches the driver program.
* The driver program asks for the resources to the cluster manager that we need to launch executors.
* The cluster manager launches executors on behalf of the driver program.
* The driver process runs with the help of user application. Based on the [**actions and transformation on RDDs**](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/), the driver sends work to executors in the form of tasks.
* The executors process the task and the result sends back to the driver through the cluster manager.
* ---------------🡪 pyspark --master yarn --conf spark.ui.port=12901 --num-executors 2
* object sc it is a spark context
* \*\*orderItems = sc.textFile(“Hdfs path”)
* In this case spark integrated with hadoop give hdfs path
* pyspark --master yarn
* 🡪we have used yarn. so it will be running in yarn mode one of the execution modes of spark
* --num-executors 2
* 🡪their are 2 worker nodes and 2 executers
* Driver is not executer. actual process data in this driver
* executor is nothing but a JVM
* sparkContext:
* which Is nothing but a web service which will manage those resources what ever created for the session..each web service run on a different port number (10000 to 65535)
* [abhilashkalva@gw02 conf]$ pyspark --master yarn --conf spark.ui.port=12901 …..
* Is a Driver program when it is launched it has created the sparkContext. sparkContext it actually Procured some resources from the clusters by default 2 executors and then sparkContext will keep track of it.
* >>> sc.setLogLevel("INFO")
* Remove log level info use
* >>> sc.setLogLevel("ERROR")

Transformations And Actions :-

* Transformations : which take a RDD and return another RDD as output.

Input is an RDD and out put also an RDD. these APIs does not trigger execution but update the DAG.

* Row level Transformations - map, flatMap, filter
* Joins – join,leftOuterJoin, rightOuterJoin
* Aggregations – reduceByKey, aggregateByKey
* Sorting data – sortByKey
* Group operations such as groupByKey
* Set operations – union, intersection
* And more
* Actions:- Actions take RDD as input and return a primitive data type or regular collection to the driver program. also we can use actions to save the output to the files. Actions triggers execution of DAG.
* Previewing data – first, take, takeSample
* Converting RDD into typical collections – collect
* Total Aggregations – count, reduce
* Total ranking – top
* Saving files – saveAsTextFile, saveAsNewAPIHadoopFile etc
* And more

Join Syntax :

ordersJoin = orderFilterdMap.join(orderItemsMap)

In Scala --- lines.toDebugString 🡪 we can see DAG level details

sortByKey :

orders.sortByKey() – ascending

orders.sortByKey(False) – descending

Directed Acyclic Graph(DAG) and Lazy Evaluation:-

There are many APIs in spark. But most of the APIs do not trigger execution of spark job.

* When we create a spark Context object it will procure resources in the cluster
* APIs used to read the data such as textFile as well as to process the data such as map ,reduce, filter etc does not trigger immediate execution. They create variable of type RDD which also point to DAG.
* They run in Driver program and build DAG. Dag will tell how it should execute. Each variable have a DAG associated with it.
* When APIs which are categorized as action (such as take, collect, saveAsTextFile) are used Dag associated with the variable executed.
* In Scala, we can look at Dag details by using toDebugString on top the variable created.
* We can visualize the DAG as part of spark.

Partition data:

* orderItems.repartition(4).saveAsTextFile("/user/abhilashkalva/pyspark/orderItemsPartitioned")
* Data is partition into 4 buckets

Whatever logic we passed to the lambda function it has to be python logic only. We cannot use spark APIs as part of the lambda function

As part of lambda functions which are passed to APIs such as map, flatMap, filter etc – the logic should be pure python

# filter, join, aggregation, sorting