**UNIT-5**

**SPARK**

Introduction to data analytics with Spark, Programming with RDDS,

Working with key/value pairs, advanced spark programming.

**Q1) What is Spark. Explain features of Spark.**

\* **Apache Spark is a powerful open source engine. Since, it offers real-**

**time stream processing, interactive processing, graph processing, in-**

**memory processing as well as batch processing.**

\* Spark was introduced by Apache Software Foundation for speeding up the

Hadoop computational computing software process.

\* Spark uses Hadoop in two ways – one is storage and second is processing.

Since Spark has its own cluster management computation, it uses Hadoop

for storage purpose only.

**Features of Spark:**

**1. Swift Processing :** Apache Spark offers high data processing speed. That

is about 100x faster in memory and 10x faster on the disk. However, it is

only possible by reducing the number of read-write to disk.

**2. Dynamic in Nature:** Basically, it is possible to develop a parallel

application in Spark. Since there are 80 high-level operators available in

Apache Spark.

3. **In-Memory Computation in Spark:** The increase in processing speed is

possible due to in-memory processing. It enhances the processing speed.

**4. Reusability:** We can easily reuse spark code for batch-processing or join

stream against historical data. Also to run ad-hoc queries on stream state.

**5. Spark Fault Tolerance:** Spark offers fault tolerance. It is possible

through Spark’s core abstraction-RDD.

**6. Real-Time Stream Processing:** We can do real-time stream processing in

Spark. Basically, Hadoop does not support real-time processing.

**7. Lazy Evaluation in Spark:** All the transformations we make in Spark RDD

are Lazy in nature that is it does not give the result right away rather a new

RDD is formed from the existing one. Thus, this increases the efficiency of the

system.

**8.Polyglot:** Spark provides high-level APIs in Java, Scala, Python, and R.

Spark code can be written in any of these four languages. It also provides a

shell in Scala and Python.

**9. Support for Sophisticated Analysis:** There are dedicated tools in Apache

Spark. Such as for streaming data interactive/declarative queries, machine

learning which add-on to map and reduce.

**10. Integrated with Hadoop:** As we know Spark is flexible. It can run

independently and also on Hadoop YARN Cluster Manager. Even it can read

existing Hadoop data.

**11. Spark GraphX:** In Spark, a component for graph and graph-parallel

computation, we have GraphX.

**12. Cost Efficient:** For Big data problem as in Hadoop, a large amount of

storage and the large data center is required during replication. Hence, Spark

programming turns out to be a cost-effective solution.

**Q2) Explain Spark Architecture in-detail.**

Apache Spark Architecture is based on two main abstractions-

• Resilient Distributed Datasets (RDD)

• Directed Acyclic Graph (DAG)

**RDDs** are the building blocks of any Spark application. RDDs Stands for:

• **Resilient:** Fault tolerant and is capable of rebuilding data on failure

• **Distributed:** Distributed data among the multiple nodes in a cluster

• **Dataset: Collection** of partitioned data with values

**Directed Acyclic Graph (DAG)** *Direct - Transformation is an action which transitions data partition state from*

*A to B.*

*Acyclic -Transformation cannot return to the older partition*

DAG is a sequence of computations performed on data where each node is

an RDD partition and edge is a transformation on top of data. The DAG

abstraction helps eliminate the Hadoop MapReduce multistage execution

model and provides performance enhancements over Hadoop.

**Fig: Spark Architecture**

Apache Spark follows master/slave architecture with two main daemons

and a cluster manager –

i. Master Daemon – (Master/Driver Process)

ii. Worker Daemon –(Slave Process/Executor)

A spark cluster has a single Master and any number of Slaves/Workers.

**Spark Driver – Master Node of a Spark Application(Master):**

It is the central point and the entry point of the Spark Shell (Scala, Python,

and R).

The driver program runs the main () function of the application and is the

place where the **Spark Context** is created. It is similar to your database

connection.

Spark Driver contains various components – DAGScheduler, TaskScheduler,

BackendScheduler and BlockManager responsible for the translation of

spark user code into actual spark jobs executed on the cluster.

**Executor/Worker Node(Slave):**

Executor is a distributed agent responsible for the execution of tasks. Every

spark applications has its own executor process.

Executors usually run for the entire lifetime of a Spark application and this

phenomenon is known as ―Static Allocation of Executors.

However, users can also opt for dynamic allocations of executors wherein

they can add or remove spark executors dynamically to match with the

overall workload.

• Executor performs all the data processing.

• Reads from and Writes data to external sources.

• Executor stores the computation results data in-memory, cache or on hard

disk drives.

• Interacts with the storage systems.

**Cluster Manager:**

An external service responsible for acquiring resources on the spark cluster

and allocating them to a spark job.

A Spark application can leverage for the allocation and deallocation of

various physical resources such as memory for client spark jobs, CPU

memory, etc.

Hadoop YARN, Apache Mesos or the simple standalone spark cluster

manager either of them can be launched on-premise or in the cloud for a

spark application to run.

**Application:**

When a client submits a spark user application code, the driver implicitly

converts the code containing transformations and actions into a logical

directed acyclic graph (DAG).

At this stage, the driver program also performs certain optimizations like

pipelining transformations and then it converts the logical DAG into

physical execution plan with set of stages.

After creating the physical execution plan, it creates small physical

execution units referred to as tasks under each stage. Then tasks are

bundled to be sent to the Spark Cluster.

spark-submit is the single script used to submit a spark program and

launches the application on the cluster.

**Q3) Write a brief note on: Spark Unified Stack.**

The Spark project contains multiple closely integrated components. At its

core, Spark is a ―computational engine‖ that is responsible for scheduling,

distributing, and monitoring applications consisting of many computational

tasks across many worker machines, or a *computing cluster*.

**Fig. *The Spark stack***

**Spark Core**

Spark Core contains the basic functionality of Spark, including components

for task scheduling, memory management, fault recovery, interacting with

storage systems, and more.

Spark Core is also home to the API that defines *resilient distributed*

*datasets* (RDDs), which are Spark’s main programming abstraction.

Spark Core provides many APIs for building and manipulating these

collections.

**Spark SQL**

Spark SQL was added to Spark in version 1.0.

Spark SQL is Spark’s package for working with structured data. It allows

querying data via SQL as well as the Apache Hive variant of SQL—called the

Hive Query Language (HQL)—and it supports many sources of data,

including Hive tables, Parquet, and JSON.

Spark SQL allows developers to intermix SQL queries with the programmatic

data manipulations supported by RDDs in Python, Java, and Scala, all

within a single application, thus combining SQL with complex analytics.

**Spark Streaming**

Spark Streaming is a Spark component that enables processing of live

streams of data.

Eg. logfiles generated by production web servers.

Spark Streaming was designed to provide the same degree of fault tolerance,

throughput, and scalability as Spark Core.

**MLlib**

Spark comes with a library containing common machine learning (ML)

functionality, called MLlib.

MLlib provides multiple types of machine learning algorithms, including

classification, regression, clustering, and collaborative filtering, as well as

supporting functionality such as model evaluation and data import.

**GraphX**

GraphX is a library for manipulating graphs (e.g., a social network’s friend

graph) and performing graph-parallel computations.

Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API,

allowing us to create a directed graph with arbitrary properties attached to

each vertex and edge.

GraphX also provides various operators for manipulating graphs (e.g.,

subgraph and mapVertices) and a library of common graph algorithms (e.g.,

PageRank and triangle counting).

**Cluster Managers**

Spark is designed to efficiently scale up from one to many thousands of

compute nodes.

To achieve this while maximizing flexibility, Spark can run over a variety of

*cluster managers*, including Hadoop YARN, Apache Mesos, and a simple

cluster manager included in Spark itself called the Standalone Scheduler.

**Q4) What is RDD? Explain the features of RDD.**

RDDs are the building blocks of any Spark application. RDDs Stands for:

• **Resilient:** Fault tolerant and is capable of rebuilding data on failure

• **Distributed:** Distributed data among the multiple nodes in a cluster

• **Dataset:** Collection of partitioned data with values

RDDs represent a collection of items distributed cross many compute nodes that can be manipulated in parallel. Basically, there are 2 ways to create Spark RDDs

**1. Parallelized collections**

By invoking parallelize method in the driver program, we can create

parallelized collections.

Eg.

>>> nums = sc.parallelize([1, 2, 3, 4])

**>>>nums.collect() [1, 2, 3, 4]**

**2. 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.

**Eg.**

lines = sc.textFile("README.txt")

There are various advantages/features of using RDD. Some of them are

**1. In-memory computation:** Basically, while storing data in RDD, data is

stored in memory for as long as you want to store. It improves the

performance by an order of magnitudes by keeping the data in memory.

**2. Lazy Evaluation:** Spark Lazy Evaluation means the data inside RDDs are

not evaluated on the go. Basically, only after an action triggers all the

changes or the computation is performed. Therefore, it limits how much

work it has to do.

**3. Fault Tolerance:** If any worker node fails, by using lineage of operations,

we can re-compute the lost partition of RDD from the original one. Hence, it

is possible to recover lost data easily.

**4. Immutability:** Immutability means once we create an RDD, we can not

manipulate it. Moreover, we can create a new RDD by performing any

transformation. Also, we achieve consistency through immutability.

**5. Persistence:** In in-memory, we can store the frequently used RDD. Also,

we can retrieve them directly from memory without going to disk. It results

in the speed of the execution. Moreover, we can perform multiple operations

on the same data. It is only possible by storing the data explicitly in memory

by calling persist() or cache() function.

**6. Partitioning:** Basically, RDD partition the records logically. Also,

distributes the data across various nodes in the cluster. Moreover, the

logical divisions are only for processing and internally it has no division.

Hence, it provides parallelism.

**7. Parallel:** While we talk about parallel processing, RDD processes the data

parallelly over the cluster.

**Fig. RDD Features**

**8. Location-Stickiness:** To compute partitions, RDDs are capable of

defining placement preference. Moreover, placement preference refers to

information about the location of RDD. Although, the DAGScheduler places

the partitions in such a way that task is close to data as much as possible.

Moreover, it speeds up computation.

**9. Coarse-grained Operation:** Generally, we apply coarse-grained

transformations to Spark RDD. It means the operation applies to the whole

dataset not on the single element in the data set of RDD in Spark.

**10. No limitation:** There are no limitations to use the number of Spark

RDD. We can use any no. of RDDs. Basically, the limit depends on the size

of disk and memory.

**Q5) Define RDD. Explain the workflow of RDD. Explain Transformations**

**and Actions on RDD.**

RDDs are the building blocks of any Spark application. RDDs Stands for:

• **Resilient:** Fault tolerant and is capable of rebuilding data on failure

• **Distributed:** Distributed data among the multiple nodes in a cluster

• **Dataset:** Collection of partitioned data with values

Basically, there are 2 ways to create Spark RDDs:

**1. Parallelized collections**

By invoking parallelize method in the driver program, we can create

parallelized collections.

>>> nums = sc.parallelize([1, 2, 3, 4])

**>>>nums.collect()**

**[1, 2, 3, 4]**

**2. 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. Eg. lines =

sc.textFile("README.txt")

**Fig. Workflow of RDD**

With RDDs, you can perform two types of operations:

1. **Transformations:** They are the operations that are applied to create a new

RDD.

2. **Actions:** They are applied on an RDD to instruct Apache Spark to apply

computation and pass the result back to the driver.

To summarize, every Spark program and shell session will work as follows:

1. Create some input RDDs from external data.

2. Transform them to define new RDDs using transformations like filter().

3. Ask Spark to persist() any intermediate RDDs that will need to be reused.

4. Launch actions such as count() and first() to kick off a parallel

computation, which is then optimized and executed by Spark.

**Actions:**

The most common action on basic RDDs you will likely use is reduce(),

which takes a function that operates on two elements of the type in your

RDD and returns a new element of the same type.

A simple example of such a function is +, which we can use to sum our

RDD.

**Eg.**

*reduce() in Python*

>>> sum = nums.reduce(lambda x, y: x + y)

>>> print sum

10

***Q6) Explain the difference between map() and flatmap()***

The map() transformation takes in a function and applies it to each

element in the RDD with the result of the function being the new

value of each element in the resulting RDD.

***Eg.***

***Python squaring the values in an RDD***

>>> nums = sc.parallelize([1, 2, 3, 4])

>>> squared = nums.map(lambda x: x \* x).collect()

>>> print squared

[1, 4, 9, 16]

**Fig. *Difference between flatMap() and map() on an RDD***

The flatMap() transformation takes in a function and applies it to each

element in the RDD *and return an RDD of the contents of the iterators*

*returned. Often used to extract words.*

Eg. ***flatMap() in Python, splitting lines into words***

>>> lines = sc.parallelize(["hello world", "hi"])

>>> words = lines.flatMap(lambda line: line.split(""))

>>> words.first()

'hello'

**Q7) Explain about Paired RDD operations.**

Paired RDD is a distributed collection of data with the key-value pair.

Transformations on Pair RDDs:

Since pair RDDs contain tuples, we need to pass functions that operate on tuples rather than on individual elements. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)}) Eg. >>>rdd = sc.parallelize({(1, 2), (3, 4), (3, 6}) >>>other = sc.parallelize({(3,9)}) >>>print rdd.collect() [(1,2),(3,4),(3,6)] >>>print other.collect() [(3,9)]

Eg.

*Word count in Python* rdd = sc.textFile("s3://...") words = rdd.flatMap(**lambda** x: x.split("")) result = words.map(**lambda** x: (x, 1)).reduceByKey(**lambda** x, y: x + y)

**Actions Available on Pair RDDs *Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))***

**Q8) What is spark? State the advantages of using Apache spark over Hadoop MapReduce for Big data processing.**

**Key Features Apache Spark Hadoop MapReduce**

Ease of Easy to code Difficult to code

Programming

Absraction Uses RDD abstraction No Abstraction

Speed 10–100 times faster than Slower

MapReduce

Analytics Supports streaming, Comprises simple Map and

Machine Learning, Reduce tasks

complex analytics, etc.

Suitable for Real-time streaming Batch processing

Complexity Easy to write and debug Difficult to write and debug

Processing In-memory Local disk

Location

Developed using Scala Java

the laguage

Supported Python,Scala,Java,R,SQL Java,Python,Ruby,Perl,C,C++

Languages

Cost High because of huge Less cost

amount of RAM

Security Evolving High compared to Spark

Coding Less no.of lines More no.of lines

SQL Through Spark SQL Through HiveQL

**Q9) Explain various statistical operation on RDDs.**

Spark provides several descriptive statistics operations on RDDs containing numeric data. count() Number of elements in the RDD mean() Average of the elements sum() Total max() Maximum value min() Minimum value variance() Variance of the elements sampleVariance() Variance of the elements, computed for a sample stdev() Standard deviation sampleStdev() sample standard deviation **Eg.1.**

>>> nums = sc.parallelize([1, 2, 3, 4]) >>> nums.count() 4 >>> nums.mean() 2.5

**Eg. 2. Removing outliers in Python**

>>> distanceNumerics = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1000]) >>> stats = distanceNumerics.stats() >>> stddev = stats.stdev() >>> mean = stats.mean() >>> reasonableDistances = distanceNumerics.filter( lambda x: math.fabs(x - mean) < 3 \* stddev) >>> print reasonableDistances.collect() [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**Q10) Write a program for wordcount in spark.**

**Wordcount.py**

import pyspark

import random

if not 'sc' in globals():

sc = pyspark.SparkContext()

text\_file = sc.textFile("/home/hadoop/Desktop/dept.txt") counts

= text\_file.flatMap(lambda line: line.split(",")) \ .map(lambda

word: (word, 1)) .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("/home/hadoop/Desktop/word")

**Q11) Write a program to calculate pi value in spark**

**pi.py**

import pyspark

import random

if not 'sc' in globals():

sc = pyspark.SparkContext()

NUM\_SAMPLES = 10000

def sample(p):

x,y = random.random(),random.random()

return 1 if x\*x + y\*y < 1 else 0

count = sc.parallelize(xrange(0, NUM\_SAMPLES)).map(sample)

.reduce(lambda a, b: a + b)

print "Pi is roughly %f" % (4.0 \* count / NUM\_SAMPLES)

**Q12) Write a program to sort given dataset in spark using data frames.**

**Sort\_rdd.py**

import pyspark

import random

from pyspark import SparkContext

from pyspark.sql import SparkSession

if not 'sc' in globals():

sc = pyspark.SparkContext()

csv\_file\_path = 'emp.csv'

employee\_rdd = sc.textFile(csv\_file\_path).map(lambda line: line.split(','))

print(type(employee\_rdd))

employee\_rdd\_sorted = employee\_rdd.sortByKey(ascending= False)

spark = SparkSession(sc)

employee\_df = employee\_rdd.toDF(['id''dept','sal'])

employee\_df\_sorted = employee\_rdd\_sorted.toDF(['sal'])

employee\_df\_sorted.show()