Scala :

Why scala is pure OOP?

Answer 1 :

yes, Scala is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes and traits. Classes are extended by subclassing and a flexible mixin-based composition mechanism as a clean replacement for multiple inheritance. . but
Java is not a pure Object oriented language, but so called a "Hybrid" language.
For any language to be pure object oriented it must follow these 6 points strictly...
1) It must have full support for Encapsulation and Abstraction
2) It must support Inheritance
3) It must support Polymorphism
4) All predefined types must be Objects
5) All user defined types must be Objects
6) Lastly, all operations performed on objects must be only through methods exposed at the objects.

Now, java supports 1, 2, 3 & 5 but fails to support 4 & 6.

Answer 2 :

https://blog.knoldus.com/is-scala-pure-object-oriented-programming-language/

Most of the people have question "Is scala pure object oriented programming language??" if yes then what about functions and primitives.
yes scala is pure object oriented language and functions and primitives are also objects in scala.

Functions as an object.

Internally function is nothing its a trait with apply function. Let see a Function1 trait with apply function as follows

```
trait Function1[A,B]{
def apply(x:A):B
}
```

Above trait shown its take one parameter as type A and return value of type B so its make a function of type **Int => Int = <function1>**

if we make a annoynomous function as below

```
val f=(x:Int)=>x*x
```

internally it would be

```
val f=new Function1[Int,Int]{
def apply(x:Int)=x*x
}
```

And when we call the function internally its call apply function of the trait

Like,if we call above function like below

```
f(4)
```

internally it will call apply function of the trait

```
f.apply(4)
```

and when we make a method it is not a function value internally it will be converted into anonymous function of type Function1,Function2 traits and expand as follows

```
new Function1[Int,Int]{
def apply(x:Int)=f(x)
}
```

The conversion of method to anonymous function this expansion expansion is known as eta expansion

Primitives as a Function

Internally all primitives are also classes which have methods for operators which is apply on object of that class type let see example of Boolean type

```
abstract class Boolean{
def ifThenElse[T](t: =>T,e: =>T):T
def &&(x: =>Boolean):Boolean=ifThenElse(x,false)
}
```

Internally true and false are object who overload the behaviour of methods.

```
object true extends Boolean
{
def ifThenElse[T](t: =>T,e: =>T)=t
}
```

```
object false extends Boolean
{
def ifThenElse[T](t: =>T,e: =>T)=e
}
```

In scala every primitive is object and operators are function so we can write 3+3 object oriented translation is 3.+(3)
Here,3's are objects and + is function class specification Int as follows

```
class Int{
def +(that:Int):Int
def -(that:Int):Int
def /(that:Int:Int
def *(that:Int):Int
def
}
```

What is functional programming?

Functional programming is a programming paradigm that uses functions as the central building block of programs. In functional programming, we strive to use pure functions and immutable values.

2.1. Immutability

Immutability means programming using constants, which means the value or state of variables can't be changed. The same goes with objects: We can create a new object, but we cannot modify the existing object's state. Thus, immutable objects are more thread-safe than mutable objects. We'll study how Scala enforces immutability in a later section.

2.2. Pure Functions

A pure function has two key properties:

It always returns the same value for the same inputs.

It has no side effects. A function with no side effects does nothing other than simply return a result. Any function that interacts with the state of the program can cause side effects. The state of the program can be mutable objects, global variables, or I/O operations, for example.

https://www.baeldung.com/scala/functional-programming

3. Case class in scala?

A Scala Case Class is like a regular class, except it is good for modeling immutable data. It also serves useful in pattern matching, such a class has a default apply() method which handles object construction. A scala case class also has all vals, which means they are immutable.

The following are some of the characteristics of a Scala case class:

- Instances of the class can be created without the new keyword.
- As part of the case classes, Scala automatically generates methods such as equals(), hashcode(), and toString().
- Scala generates accessor methods for all constructor arguments for a case class.

```
case class Student(name:String, age:Int)

object MainObject

{

  def main(args:Array[String])

  {

    var c = Student("Aarav", 23)

    println("Student name:" + c.name);

    println("Student age: " + c.age);
    }
}
```

Tuple in scala?

https://www.baeldung.com/scala/tuples

Sometimes we need a way to group several elements together that may or may not be related. For this very scenario, we can use a tuple, which simply put is an ordered collection of objects of different types.

Tuples can be particularly useful when we need to return multiple values from a function or pass multiple values to a function.

Creating a tuple is very easy and doesn't require any boilerplate code. **It's just a matter of enclosing its elements in parentheses**:

val tuple: (**String**, **Int**) = ("Joe", 34)

In Scala, tuples are defined using a series of classes named *Tuple2*, *Tuple3* all the way throug*h* to *Tuple22*.

Option in scala?

http://www.bigdatainterview.com/option-some-none-in-scala-or-how-to-handle-null-values-in-scala/

Functional programming is like writing a series of algebraic equations, and because you don't use null values in algebra, you don't use null values in FP. Scala's solution to handle null values is to use constructs like the Option/Some/None classes.

Imagine that you want to write a method to make it easy to convert strings to integer values, and you want an elegant way to handle the exceptions that can be thrown when your method gets a string like "foo" instead of something that converts to a number, like "1". A first guess at such a function might look like this:

```
def toInt(s: String): Int = {
    try {
        Integer.parseInt(s.trim)
    } catch {
        case e: Exception => 0
    }
}
```

The idea of this function is that if a string converts to an integer, you return the converted Int, but if the conversion fails you return 0. This might be okay for some purposes, but it's not really accurate.

Using Option/Some/None:

Scala's solution to this problem is to use a trio of classes known as Option, Some, and None. The Some and None classes are subclasses of Option, so the solution works like this:

- You declare that toInt returns an Option type
- If toInt receives a string it *can* convert to an Int, you wrap the Int inside of a Some
- If toInt receives a string it *can't* convert, it returns a None

The implementation of the solution looks like this:

```
def toInt(s: String): Option[Int] = {

  try {

    Some(Integer.parseInt(s.trim))

  } catch {

    case e: Exception => None

  }

}
```

This code can be read as, "When the given string converts to an integer, return the integer wrapped in a Some wrapper, such as Some(1). When the string can't be converted to an integer, return a None value."

```
scala> val a = toInt("1")

a: Option[Int] = Some(1)


scala> val a = toInt("foo")

a: Option[Int] = None
```

Higher order functions in scala?

https://www.baeldung.com/scala/higher-order-functions

http://www.bigdatainterview.com/higher-order-functions-in-scala/

Higher Order functions take other functions as parameters and return function as result, i.e., passing functions as parameters to other functions and to get a new function as result. The best example of Higher Order functions in Scala is **map().**

This concept will be clear if we look at some example code snippets explaining it.

```
object HigherOrder{
  def operation(x: Int, y: Int, func: (Int, Int) => Int): Int = func(x,y)


  def main(args: Array[String]){
    var output = operation(10,20, (x,y)=>x+y)
    println(output) //Prints sum of 10,20 i.e., 30


    output = operation(10,20, (x,y)=>x*y)
    println(output) //Prints multiplication of 10,20 i.e., 200
  }
}
```

In the above example we have defined a function with name operation, which takes two Int variable and a function that takes two integers and returns an integer.

After that we have defined main function in which we have called the function operation by passing 10 and 20 as first two parameters and an anonymous function which returns the sum of the two elements passed to it. Printing this gives us the output 30.

We have again called the same function with the 10 and 20 as first two parameters but the function passed is multiplication. Which gives us the output 200.

Closure in scala?

https://www.learningjournal.guru/article/scala/functional-programming/closures/

Scala closures are functions whose return value is dependent on one or more free variables declared outside the closure function. Neither of these free variables is defined in the function nor is it used as a parameter, nor is it bound to a function with valid values. Based on the values of the most recent free variables, the closing function is evaluated.

```
var p = 10
def getHike(salary: Double) = salary * p / 100
getHike(5000)
```

Traits in scala?

https://www.baeldung.com/scala/traits

The concept of traits is similar to an interface in Java, but they are even more powerful since they let you implement members. It is composed of both abstract and non-abstract methods, and it features a wide range of fields as its members. Traits can either contain all abstract methods or a mixture of abstract and non-abstract methods. In computing, a trait is defined as a unit that encapsulates the method and its variables or fields. Furthermore, Scala allows partial implementation of traits, but no constructor parameters may be included in traits. To create traits, use the trait keyword.

```
trait MyCompany
{
  def company
  def position
}
class MyClass extends MyCompany
{
  def company()
  {
    println("Company: InterviewBit")
  }
  def position()
  {
    println("Position: SoftwareDeveloper")
  }
  def employee()              //Implementation of class method
  {
    println("Employee: Aarav")
  }
}
object Main
{
  def main(args: Array[String])
  {
    val obj = new MyClass();
    obj.company();
    obj.position();
    Obj.employee();
   }
}
```

Function currying in scala?

Currying in Scala is a technique of transforming a function that takes multiple arguments into a function that takes only one argument. To achieve this we will convert the function having list of parameters to a function with chain of functions one for each parameter.

Let's take a look at some code snippets of Currying functions.

Below is a code snippet that done without using Currying:

```
object CurryingFunctions {
  def addition (a: Int, b: Int) = a + b
  def main(args: Array[String]) {
    println(addition(4,5))
  }
}
```

Below is another code of the same addition function with Currying:

```
object CurryingFunctions{
  def addition (a: Int) (b: Int) = a + b; //Currying


  def main(args: Array[String]) {
    println(addition(4)(5));
  }
}
```

The same function can be called using Partially applied functions:

Partially applied function means applying some of the parameters of function. Then we will get a partially applied function that can be used later.

```
object CurryingFunctions {
  def addition(a: Int) (b: Int) = a + b;

  def main(args: Array[String]) {

    val total = addition(5)_ //Partially applied function

    println(total(10)) //Output: 15

  }

}
```

Tail recursion in scala?

https://blog.knoldus.com/recursion-tail-recursion-in-scala/

Recursion is a function which calls itself over and over again until an exit condition is met. It breaks down a large problem into a smaller problem, solves the smaller parts, and combines them to produce a solution. Recursion could be applied to many common problems, which can be solved using loops, for and while.

Why Recursion?

1. The code is simpler and shorter than an iterative code as we only need to define the base case and recursive case.

2. It avoids the mutable variables that you need to use while writing loops.

Problem with Recursion

1. For every recursive call, separate memory is allocated for the variables.
2. Recursive functions often throw a Stack Overflow Exception when processing or operations are too large.

Why Stack Overflow Error occurs?

In a recursive function, every method call will create its stack frame in the stack memory. The first call of the function doesn't return until the last call in the recursion is resolved. So, the function return order is from last invoked to first invoked. This means that if the function F calls itself recursively for N times, then it would create N stack frames in the stack memory for a single execution of the function F. The compiler will keep all the N stack frames in memory till the recursion is complete.

To solve the stack overflow error we can use tail-recursion.

Tail Recursion

A recursive function is said to be tail-recursive if the recursive call is the last operation performed by the function. There is no need to keep a record of the previous state. In Scala, you can use @tailrec to check if the recursion is tail-recursive or not. The annotation is available in the scala.annotation._ package. If the recursion isn't tail-recursive, then Scala will throw a compile-time error.
When the Scala compiler recognizes that it is tail-recursion, it will optimize the function by converting it to a loop. We will not realize the change, but the compiler will do it internally. This optimization will overcome the performance and memory problem.

Example : sum of n number using recursion :

```
Def sum( num : Int) : Int ={
        If(num == 1) return 1
        Else sum(num-1) + num
}
```

Example : sum of n number using head recursion :

```
Def sum( num : Int, res : Int) : Int ={
        If(num == 1) return res
        Else sum(num-1 , res + num)
}
```

What is Singleton object?

Scala doesn't have a concept called **static**. Instead of static, scala has something called **Singleton** object. A singleton object is an object that defines only one instance of a class. Normally this singleton object provides entry point any program we write in Scala. Without this singleton object we can compile a Scala program but we cannot get output of it. Singleton object will be defined with a preceding keyword **object** before the object name.

Syntax:

```
object Name{
// code...
}
```

Properties of Singleton object:

- The method in the singleton object is globally accessible.
- You are not allowed to create an instance of singleton object.
- You are not allowed to pass parameter in the primary constructor of singleton object.
- In Scala, a singleton object can extend class and traits.
- In Scala, a main method is always present in singleton object.
- The method in the singleton object is accessed with the name of the object(just like calling static method in Java), so there is no need to create an object to access this method.

Scala Companion Object

A companion is an object that has exact name as a class. Then we will it as companion object of that class. Companion object and it's class must be in same source code file. Both class and it's companion object can access the other's private variables.

Code:

```
object CompanionObjectDemo {

def main(args: Array[String]):Unit = {

val compObj = new CompanionObjectDemo()

compObj.companionObjectMethod

compObj.privateMethod

}

}


class CompanionObjectDemo{

def companionObjectMethod():Unit = {

println("Companion object's method called")

}

private def privateMethod():Unit = {

println("Even private methods can be accessed using compnion object")

}

}
```

Output

```
Companion object's method called

Even private methods can be accessed using compnion object
```

Constructors in scala?

https://www.geeksforgeeks.org/scala-constructors/

inheritance in scala?

https://blog.knoldus.com/inheritanceand-its-types-in-scala/

SPARK

Spark Architecture and What happens when you perform spark submit?

http://www.bigdatainterview.com/explain-spark-architecture-or-what-happens-when-submit-a-spark-job/

This is one of the most common interview questions. As a Spark applications developer we must have the idea of the architecture of Spark. Here's the blog to get complete knowledge on Spark Run Time Architecture.

In this blog we will get to know the run-time architecture of Spark along with some key concepts in it and terminologies like SparkContext, Shell, Application submission, task, job, stage etc. Ans also the driver program role, executors role and cluster manager role.

Spark Architecture is bases on two main abstractions. They are:

1). RDD (Resilient Distributed Dataset)

2). DAG (Directed Acyclic Graph)

RDD - An RDD is a collection of items that are divided into partitions. These partitions are stored in memory of the work nodes in Spark cluster. RDD's can be created using the files on HDFS or any other storage system supported Spark and using parallelized collections of Scala native collections. RDD supports two types of operations: 1). Transformations 2). Actions

DAG - DAG is graph that stores series of operations as edges and rdd's as nodes. The operations that are mentioned on the edges will be carried on the corresponding nodes, which are RDD's.

Daemon processes in Spark

Spark has master-slave architecture using two main daemon processes:

i). Master Daemon

ii). Worker Daemon

In Spark only one master will be there and multiple workers. Driver program is the master of all the executors. Driver and executors will run on their on JVMs.

Before getting deeply diving into architecture we must have known some terms and their definitions. They are:

SparkSession/SparkContext -

This is the heart of Spark application execution. By using this we can create RDDs, accumulators, broadcast variables to execute the program. The main duties of SparkContext are:

- To keep track of status of an application
- To cancel a job
- To cancel a Stage
- Running the jobs synchronously or asynchronously
- Dynamic resources allocation
- Accessing persisted data and to unpersist the persisted data

Spark -Shell

This is an application developed in Scala, which offers the user an interactive command line environment with auto-completion of statements. This can be used to get familiar with the functionalities available in Spark and to run small sample programs there by getting knowledge to develop a complete standalone job.

Task

Task is a unit of work that will run on the executor. Each stage will have some task and one task will be running on one partition. The same will run on the all the partitions on all the executors.

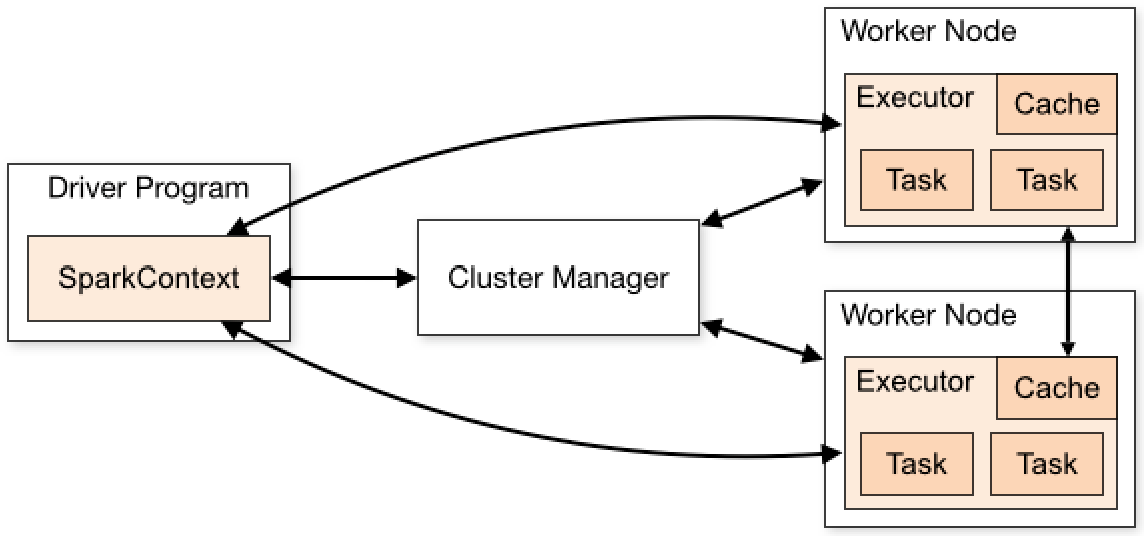
Job

Job is a set of tasks that will be running in parallel. They are the responses to the actions function of Spark.

Stage

Each job is divided into sets of tasks called Stages. Stages will depend on each other i.e., later stage will depend on the former stage. An application process will be done in multiple stages.

Below is the diagram that shows the architecture of Spark:



Let's look into each and every component present in the above architecture diagram:

Driver Program

This is the entry point to a Spark job. The **main()** method of the Spark job will be defined in driver program and this is where the SparkContext or SparkSession will be created. The driver program runs the user code on executors and creates RDDs and performs the transformations and actions. It divides the Spark applications into Stages and schedules them for running on the executors. The task scheduler always lies in the driver program and distributes the program among the worker nodes or executors.

Cluster Manager

This is the one which allocate the memory and resources for the driver program to execute. The resources that are required for job can be requested dynamically whenever required and will bee freed whenever they are not getting used. For client mode this might not present but in production environment client mode never be used, so this is an optional component. There are several cluster managers available that work with Spark. They are: 1). Standalone, 2). YARN 3). Mesos 4). Kubernetes. Choosing one among these cluster mangers depends on the final goal of our project purpose. Because all of these will provide different set of schedulers and scheduling capacities.

Executors

The divided tasks will be executed on the individual executors in Spark cluster. Every Spark job will have its own executors and these executors will be at the the beginning of the Spark job and will be terminated at the end of the execution. This is called Static Resources allocation. However we can configure for Dynamic Resources Allocation, in which the resources will be allocated and freed as and when required. Executors are the actual data processors. They read the data for processing and write the output to storage systems. They store the computational data in memory.

How to launch a Spark program?

We can launch a Spark program using spark-submit command. We can supply all the inputs required and additional configurations like driver memory, number cores etc in this command. This program launches the driver program to start the execution.

What happens when we submit a Spark Job?

Using spark-submit command user submits the Spark application to Spark cluster. This program invokes the **main()** method that is specified in the spark-submit command, which launches the driver program. The driver program converts the code into Directed Acyclic Graph(DAG) which will have all the RDDs and transformations to be performed on them. During this phase driver program also does some optimizations and then it converts the DAG to a physical execution plan with set of stages. After this physical plan, driver creates small execution units called tasks. Then these tasks are sent to Spark Cluster.

The driver program then talks to the cluster manager and requests for the resources for execution. Then the cluster manger launches the executors on the worker nodes. Executors will register themselves with driver program so the driver program will have the complete knowledge about the executors. Then driver program sends the tasks to the executors and starts the execution. Driver program always monitors these tasks that are running on the executors till the completion of job. When the job is completed or called stop() method in case of any failures, the driver program terminates and frees the allocated resources.

YARN ARCHITECTURE :

https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

Spark on YARN?

https://sujithjay.com/spark/with-yarn

Spark internal working?

https://stackoverflow.com/questions/30691385/how-spark-works-internally

Spark context vs Spark Session?

One of the most commonly asked interview questions. If you are mid-level experienced professional this will be compulsory question. In this blog we will get to know about these to abstractions in detail and also you will get to know some knowledge about it which is not available on most commonly used forums like stackoverflow.

In Spark version 1.x we have the SparkContext as entry point for out Spark applications, for which we need to create SparkConf object. All our configurations will be set to SparkConf object and using which SparkContext object will be created as shown below:

```
val conf = new SparkConf().setAppName(“MyAppliction”).setMaster(“local”)

val sparkContext = new SparkContext(conf)
```

Using the SparkContext above execution of the entire application starts.

Whereas in Spark version 2.x we have SparkSession as entry point. Which is completely different from SparkContext. Using this SparkSession we create DataFrames and Datasets and accomplish the task. Typical SparkSession creation code snippet looks similar to below code snippet:

```
val spark = SparkSession

        .builder

        .master("local")

        .appName(“WorldBankIndex”)

        .getOrCreate()
```

SparkContext object will also be created while creating SparkSesssion object and it can be accessed from the SparkSession created above, as shown in the below:

```
val context = spark.sparkContext
```

In addition SparkSessin object by default will contain the SQL context object also.

This is not the complete answer for our question. This is an answer that is known to naive in Spark. If you are an experienced professional, interviewer will expect something more for this question. And if you are done with the above answer during the interview and if the the interviewer is Kudoos, you will screwed there. His next question would be what is the need of doing that? We will that now..

If you observe two different SparkSessions created in an application will have the same SparkContext object. Try to create two different SparkSessions, get the SparkContext objects out of them and compare the hashcode of them. They will be same. This is because having multiple SparkContext objects in single application might lead unexpected behavior of Spark application. And also failure of one SparkContext can cause the failure of the other, which ultimately can cause JVM to stop working. It's not guarantee that having multiple SparkContext objects in application will make pipeline flow to work properly.

Another reason for having only one SparkContext object and multiple SparkSesssions is multiple users to process the data. If we are using Spark version 1.6 and we have multiple users to operate on the datasets and need the datasets to be isolated for each user then we have to create multiple SparkContext objects, which is not at all acceptable in performance perception. Whereas for the same situation is handled in Spark version 2.x by creating multiple SparkSessions and single SparkContext. So to handle this situation SparkSession was created.

https://www.youtube.com/watch?v=qQ8pidVduBA

DAG VS Lineage

https://www.youtube.com/watch?v=NGOD7JN6azM

Executor Tuning

This will be must and should question in Spark interviews. Understanding the resource allocation for a Spark job is very important, because entire Spark job execution completely depends on these parameters. If we don't allocate the resources properly job might run into issues and the might be starving for the resources. In this blog we will get the knowledge how to allocate the resources for Spark job like how many executors needed, how much memory required for each executor and how many number of cores required for each core. Below are the factors based on which we need to allocate the resources for Spark Jobs. They are:

- The size of the data we going to process
- Time within which the job needs to be completed
- Resources allocation - Dynamic/Static
- Upstream or Downstream application

Static Resources Allocation

Assume that we have 6 nodes and each node has 16 cores and 64GB RAM.

In each and every node 1 core and 1 GB RAM must be allocated for the OS and background daemon process that always run. So we are left with 15 cores and 63 GB RAM in all the nodes.

Number of cores:

Number of cores will be equal to number of concurrent tasks that run on an executor. But here is a limitation on number of cores. As discussed just now number of cores are equal to number of tasks that an executor can run but this should 5, because researches show that more than 5 parallel tasks lead bad performance of Spark job. So the optimum value to run job properly, the number of cores must be 5, even if we have 15 cores available in each node.

Number of executors:

We have 15 cores available on each node and each executor can have 5 cores. So the total number of executors can a node has is 15/5 = 3. If we calculate number of executor for all the 6 nodes it comes as, 3*6 = 18, because we have 3 executors on each node and we have 6 nodes. So we have total 18 executors out of which one executor must be dedicated to ApplicationMaster in YARN. So we are left with 17 executors. So we can pass 17 to the parameter --num-executors along with spark-submit command.

Memory for each executor:

We have 3 executors on each and 63 GB RAM on each node.

So each executor can take 63/3 = 21 GB RAM.

But there will be small memory overhead involves to determine full memory request to YARN.

So the formula for memory overhead is maximum(384MB, 0.7*spark.executor.memory).

On calculating this we will get max(384 MB, 1.47GB).

So the maximum out of these two is 1.47 GB so the memory overhead is 1.47 GB.

So the memory available for each executor will be 1.47 GB which can be rounded off to 2 GB lesser than available.

So the memory available for each executor is 19 GB.

Final resources:

--num-executors 17

--executor-cores 5

--executor-memory 19GB

https://www.youtube.com/watch?v=V9E-bWarMNw&list=PL9sbKmQTkW05mXqnq1vrrT8pCsEa53std&index=2

AVRO vs ORC vs Parquet

https://www.clairvoyant.ai/blog/big-data-file-formats

https://www.youtube.com/watch?v=q8uqpwxaVZw

https://www.youtube.com/watch?v=VQnlzBw9Pm0

What is DAG scheduler in Spark?

DAG - Directed Acyclic Graph

A DAG comprises of edges and vertices, in which edges represent rdds and vertices represent operations to be performed on rdds. On calling any action DAG will be submitted to DAGScheduler. Further this job will be divided into stages, where a stage is operations between two shuffles.

How DAGScheduler works in Spark?

i). Scala interpreter works on the code first to create binary code.

ii). Spark creates a graph after compiling the source code.

iii). Whenever an action called, Spark submits this graph to DAGScheduler.

iv). DAG scheduler divides the job into stages.

v). These stages are passed to task scheduler and through cluster manager spark launches the job.

vi). Finally worker nodes will execute the job.

How DAG achieves fault tolerance?

Spark always divides the task into stages. During the execution each executor works on one partition at a time. These operations together called a DAG. If there is any failure like node failure, Spark recognizes it. Then it first finds the missing partition, as it has all the steps in DAG which has all the steps of the job, it rebuilds the missing partition. So there won't be any data loss.

How DAG optimizes the Spark Job?

DAG optimizer does this job. DAG optimizer rearranges the operations in such way that processing time will be lesser. For example consider we created a job which has map job followed by filter operation. Then DAG scheduler rearranges filter operation first and then map. So the number of records that will operated in map operation will be lesser as some of the records are already filtered.

What is the difference between DAG and Lineage graph?

This is a common interview question that we face in interviews. DAG maintains the steps to rebuild the partitions and Lineage also does the same. Then what is the difference between these two.

Lineage a set of steps which will be used to rebuild partitions of an RDD. Lineage is confined to RDDs only. Whereas the DAG is a combination of edges and vertices. Vertices represent rdds and edges represent the operations to be performed on them. DAG always divides the task to in stages but rdd will not.

Catalyst Optimizer?

https://www.youtube.com/watch?v=J6s8PxdxKPM

Spark Memory Management?

https://www.linkedin.com/pulse/apache-spark-memory-management-deep-dive-deepak-rajak

Spark RDD vs Dataframe vs Dataset

Spark release version

RDD - RDDs are native API of Spark. They are introduced in Spark 1.0 version

Dataframes - Released in 1.3 version of Spark

Dataset - These are introduced in 1.6 version of Spark

Lazy Evaluation

RDD- RDDs use lazy evaluation. Means that they process the data when any action is performed.

Dataframe - Dataframes also uses lazy evaluation

Dataset - Datasets use lazy evaluation

Programming Language support

RDD - RDD API is available in multiple languages like Java, Scala, Python and R. Can be considered as a flexibility offered by RDDs for the developers.

Dataframe - It is also available in all the languages in which RDD is also available.

Dataset - Dataset is available currently in Java and Scala languages only. Python and R are not supporting datasets right now.

Compile-Time Safety

RDD - They provide compile time safety.

Dataframe - They provide run-time safety. If you try to access any column that is not present in dataframe, it throws error at run time.

Dataset - Dataframes provide compile time safety.

Data formats

RDD - Can work with structured and unstructured data. But it cannot have well defined schema. User need to work little more to use RDDs

Dataframe - Can work with structure and semi structured data. It can organize the data in the name columns. So it can have schema embedded within it.

Dataset - It can efficiently process structured and unstructured data. It represent data in the form of Java objects of row objects.

Data sources

RDD - RDD can be created from different sources like text file, JDBC connection etc.

Dataframe - Dataframe can be created from different file formats or from any RDD.

Dataset - Datasets also can be created from different file formats as well as from RDDs.

Optimization

RDD - No optimization engines are embedded in RDDs. RDDs won't come with any optimizers like Catalyst optimizer or Tungsten optimizer. So they give worst performance when working with structured data. Developer needs to optimize them using coding techniques.

Dataframe - It has **Catalyst Optimizer**, which is the inbuilt optimizer in Dataframes.

Dataset - Datasets also come with **Catalyst Optimizer**.

Serialization

RDD - Spark RDD distributed over the cluster in the form of Partitions. If there is any shuffling or any partition needs to be written to disk Spark uses Java Serialization and this process is very costly as it takes more time and CPU resources.

Dataframe - Spark dataframes can serialize the data into memory(off - heap memory) in binary format and can perform all the transformation operations directly in memory itself. So thee is no need of Java serialization in case of Dataframes. It uses Tungsten execution in the backend to generate the byte code for the expressions evaluated, which explicitly manages the memory.

Dataset - In case of Datasets, they will have encoders which handle the communication between JVM objects to tabular representation. Dataset stores the tabular representation using Tungsten binary format. Dataset allows the user to perform operations directly on the serialized data, which improves the memory usage.

Garbage collection

RDD - RDD has lot of memory overhead.

Dataframe - It has lesser garbage collection compared to RDD.

Dataset - There is no need of garbage collector as it Tungsten serialization, which uses off heap data serialization.

Operation fastness

RDD - RDDs will give worst performance while doing aggregations on them

Dataframes - As they have optimizers embedded in them, they give good performance and faster results when we perform aggregation operations.

Datasets - Datasets also provide faster result as they have Catalyst optimizer.

Schema inference

RDD - RDDs won't have any schema, so we can't perform operations using column names.

Dataframe - Dataframes can infer the schema from the files and users also can define a custom schema.

Dataset - Dataset also can infer the schema from the files using Spark SQL framework and user can also define the schema.

Schema Evolution in spark and hive?

https://kontext.tech/article/381/schema-merging-evolution-with-parquet-in-spark-and-hive#:~:text=With%20schema%20evolution%2C%20one%20set,schema%20of%20those%20files%20automatically.

Spark Shuffle Service?

https://www.youtube.com/watch?v=AsVYpQ2Ow4A

Spark Dynamic resource allocation ?

https://www.youtube.com/watch?v=-9bh_Oue9GM

https://www.waitingforcode.com/apache-spark/external-shuffle-service-apache-spark/read

Small file problem in HDFS.

http://www.bigdatainterview.com/what-is-the-need-of-large-block-size-in-hadoop/

Storing large numbers of small files in HDFS is called as Small File Problem in HDFS.
The word problem simply implies that Hadoop is not really designed to process large number of small files.
It doesnot work well with large no of small files.

Small Files : File which is significantly smaller than default HDFS block size.

Hadoop faces problems on below two level while dealing with small files -

1. Storing level (HDFS)

This problem occurs in HDFS at namenode.
As we know namenode stores matadata of slave directories, files and blocks.
This data is stored in volatile memory like RAM in the form of object.
Approx size required to store each object is 150 bytes.
So it takes more memory to store large no of small files.
As example, 20 million files will take 3 GB of memory.
Scaling volatile memory like RAM beyond this is problem with current hardware.

2. Processing level (Map-Reduce)
A. Disk seek issue
Large number of small files means large amount of disc IO.

Lots of time get spent in disc seek which becomes overhead on system and mitigates system performance.
B. Many map tasks due to many blocks
Mapper usually process one block of input at a time.
Due to large number of small files, a lot more mappers are required to complete job which inturn kills system's performance.

Spark performance tuning?

https://www.youtube.com/watch?v=Kb08RTmjnkw

Spark Join Types?

https://www.youtube.com/watch?v=isOuTH_49pY

Out Of Memory?

https://www.youtube.com/watch?v=RDKAk0TqJ8A

Handling Data Skew?

https://www.youtube.com/watch?v=HIlfO1pGo0w&list=PLtfmIPhU2DkNWZsSBDKclMfLU8uCOB1kW&index=11

window functions?

https://sparkbyexamples.com/spark/spark-sql-window-functions/

UDF?

https://www.youtube.com/watch?v=w3i3rRKdrm4

Parsing nested json data in spark?

https://docs.databricks.com/spark/latest/dataframes-datasets/complex-nested-data.html

Programming Question

How to handle CSV file where address column data's are seperated by comma as a single column instead of multiple column in Spark?

https://stackoverflow.com/questions/61421644/how-to-handle-csv-file-where-address-column-datas-are-seperated-by-comma-as-a-s

https://stackoverflow.com/questions/60964592/read-csv-file-with-values-containing-delimiter-in-apache-spark/60965746#60965746

remove empty lines from dataframe

https://stackoverflow.com/questions/53769387/spark-scala-how-to-remove-empty-lines-either-from-rdd-or-from-dataframe

MISC

https://www.youtube.com/watch?v=kiBvw1m5CJI&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=36

https://www.youtube.com/watch?v=InYqthDjn5I&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=37

https://www.youtube.com/watch?v=ooRSu2d7wWI&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=38

https://www.youtube.com/watch?v=MNWUQ5dGQT0&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=39

https://www.youtube.com/watch?v=XDLWgZ5GGUA&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=40

https://www.youtube.com/watch?v=AepDmIRTD9U&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=41

https://www.youtube.com/watch?v=V6TvvKSJIiY&list=PLmYUGOKB0Iqyg1eHPn1ZgH9VbDBiBc75t&index=42

https://www.youtube.com/watch?v=TY3_G4SFhGo

https://towardsdatascience.com/knowing-pyspark-and-kafka-a-100-million-events-use-case-5910159d08d7

https://developer.hpe.com/blog/performance-tuning-of-an-apache-kafkaspark-streaming-system/

https://alvintoh.gitbook.io/apache-2-0-spark-with-scala/section-3-spark-basics-and-simple-examples/14.-activity-running-the-average-friends-by-age-example

https://medium.com/swlh/building-partitions-for-processing-data-files-in-apache-spark-2ca40209c9b7#:~:text=size'%20at%20default%2C%20No.,of%20partitions%20is%20also%201023.

Avg of age by name

```
val data_rdd =  sc.parallelize(Seq(("Anand",20),("Ramesh",30),("Raja",21),("Anand",23)))
val value_map = data_rdd.mapValues(x => (x,1))

val data_red = value_map.reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))

val data_agg = data_red.map(x => (x._1, x._2._1 / x._2._2))
```

Character count in scala

```
object MyClass {

    def main(args: Array[String]) {
        val txt = "ABRACADABRA"
        val txt_split = txt.split("")
        txt_split.foreach(println)
        println("Mapped Text")
        val txt_map = txt_split.map(x => (x,1))
        txt_map.foreach(println)
        val txt_grouped = txt_map.groupBy(_._1).mapValues(alp=>{alp.map(_._2).sum})
        println("COunted Text")
        txt_grouped.foreach(println)
    }
}
```

word count in spark

```
val txt = sc.textFile("demo.txt")

val words = txt.flatMap(x=>x.split(","))

val word_kv = words.map(x => (x,1))
```

vall word_count = word_kv.reduceByKey(_+_)

Word Count Map Reduce Phases?

https://dzone.com/articles/word-count-hello-word-program-in-mapreduce

Operators in AirFlow?

https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html

SQOOP

How to set number of reducers for a Sqoop job?

http://www.bigdatainterview.com/how-to-set-number-of-reducers-for-a-sqoop-job/

How can you set number of reducers for Sqoop job?

How many reducers did you use for your Sqoop job?

How many default number of reducers will Sqoop job uses?

When you face this question during an interview and you don't know the answer, you get two questions in your mind: "Why don't I get these type questions when I'm preparing for the interview?" and "Did I ever set number of reducers for a Sqoop job?". Of course, I got the same thought. I faced this question in almost all the interviews. But don't worry. I'm here to help you with the answer and the reason for that.

Why Sqoop doesn't require reducers or reduce jobs?

Reducers are required for aggregation jobs. Sqoop never uses reducers, because it does parallel import of the data from RDBMS to Hadoop. So There won't be any aggregation operation involved.

When you tell this answer to interviewer he will be ready to shoot you with another question: "What if I specify a query in the option --query of Sqoop job which involves an aggregation?". This Sqoop job also doesn't require any reducers. Because whatever the aggregation query that is mentioned in Sqoop job, will be performed in RBMS only. After that the result data of the query will be imported to Hadoop using parallel processing.

1). How can you list the columns of a table using Sqoop?

We don't have any direct option to accomplish this task. But we have commands in SQL that can be used to do this. So below is the Sqoop command that can be used to display all the columns of a table:

Sqoop import

--connect "jdbc:sqlservername/"

--username username

--password password

--eval Query

2). Difference between --target-dir and warehouse-dir

--target-dir: This argument takes the path of a folder inside which the data files that are imported from the RDBMS tables will be stored. This is used when we import single table from database.

--warehouse-dir: This used when we import multiple tables from RDBMS. Takes path of a folder in HDFS and creates sub-folder with the names of source tables and inside those folder it stores the data files of corresponding tables.

3). What is Free Form Import in Sqoop?

It means that user can import data from any RDBMS table using any SQL query rather than using the options table name and column name provided by Sqoop.

4). How can we execute Free Form SQL query to import the rows sequentially in Sqoop?

This can be achieved by setting the --num-mappers or -m argument to 1. Which creates only one task and imports the rows sequentially into HDFS.

5). What is the default file format for importing the data from RDBMS tables?

The default file format while importing data from database is text file format. User can explicitly specify this using the argument --as-textfile.

6). What is Sqoop metastore?

Sqoop metastore is the repository of all the saved jobs that are created using sqoop job command. The configuration details of metastore will be done sqoop-site.xml file.

7). How to import all the tables of a database expcept some tables?

Assume that we have 300 tables in a database and we want to import all the 300 but not table150, table180 and table 190. How can we do this in Sqoop?

There is an option in sqoop import-all-tables command, --exclude-tables, where we can specify all names of the tables which we don't want to import.

--exclude-tables table150, table180, table190

8). During Sqoop job preparation you mentioned --num-mappers to 8, but you found that the Sqoop job runs only 4 mappers. Why?

The possible reason for this is Hadoop cluster is set to use only 4 mappers for that user. In that case Sqoop cannot use more than 4 mappers even if we pass more than 4.

9). What is the significance of --split-by clause in Apache Sqoop?

Split-by clause is used to specify the column based on whose values the splits will be created. If we don't mention this argument by default Sqoop takes primary key column for splits generation. But some times primary key column might not have evenly distributed values between the maximum and minimum ranges. So the splits will of different sizes, so the execution of mapreduces job, which be will one for each split might take different times to complete. So to improve the performance in such case we can go with split-by command by supplying a column to it in such a way that the column has evenly distributed data between the ranges.

10). Even after creating having --split-by command in Sqoop command, Sqoop job is giving optimal performance. How can we optimize this?

There will cases where even after supplying the --split-by command Sqoop job might not give good perfoemance. In such cases we can use --boundary-query which can be used to specify maximum and minimum values for split column through an SQL query.

11). Is it mandatory to have metastore in Hadoop cluster?

No. It is not mandatory to have the metastore in cluster, we can have it even outside of the cluster.

1). How to import large object such as BLOB and CLOB using Sqoop?

There are no arguments available for direct import large objects in Sqoop. So in order to import the large objects JDBC based imports have to be made without using direct arguments.

2). What is Sqoop Job?

To perform incremental updates we need to the run Sqoop job with incremental column updates. So to accomplish this we can create a Sqoop job, which is stored in metastore along with update values and we can it any time we want. This is to continuously import the data incrementally.

3). How can we update the rows which are already exported to RDBMS?

To update the already exported rows we can use the argument --update-key which takes the Sqoop job to update mode. In the argument --update-key we can mention all the columns which uniquely identifies a row. All those columns will be used in WHERE clause of the update query that will be generated when we run the job. Remaining columns will be used SET clause to update the values.

4). How to compress the output file of a Sqoop job?

There are arguments available to do compress the output. The argument --compress enables compression and the argument --compression-codec allows the user to specify the compression algorithm that is to be used with this.

5). What is Sqoop merge tool.

Sqoop merge to is for combining two datasets. We can use Sqoop merge tool to merge to files where the older file's data will be replaced with new file's data and the new records will be added.

6). What is codegen in Sqoop?

Codegen is a tool available in Sqoop to generate Java code for data records of a database.

7). How can you import a subset of based on a condition from a database?

We can use --where parameter we can do this.

Eg: --where "id_num = '12345'"

8). How to perform incremental data load in Sqoop?

Some times business might need to you process the that is daily generated along with data that is already getting processed everyday. So in such cases you need to get the latest records after the last record imported. This type of extraction is called incremental load.

To perform incremental load we need to use three arguments.

--incremental: This is to set the mode incremental loads. This will take one of the two arguments, append or lastmodified. The mode append will take latest appended records based on the value of --check-column argument.

--check-column: This argument is to check the column from which latest records to be identified.

--last-value: This argument is to specify the last imported value. Based on this values Sqoop will take next record onwards.

9).What is the default database that Apache Sqoop has?

The default database in Sqoop is MySql.

10). What is eval tool in Sqoop?

The tool eval is used to test the database connections and to execute some sample queries to check whether the desired results are coming or not.

11). How to set number of reducers for a Sqoop job?

Sqoop never uses reducers.

12). You have set number of mappers to 4 while importing data but you are not sure whether the table has primary key column or not. How can you handle this situation?

If we have set number of mappers to more than 1 while importing and the database table has no primary key column then the Sqoop job fails. To avoid this we have use --autoreset-to-one-mapper argument, which automatically resets the number of mappers to 1.

13). Difference between Sqoop and Flume (or) Sqoop vs Flume.

- Sqoop is used to move data from structured datasources like RDBMS whereas Flume is used in bulk data streaming operations
- Sqoop has connector based architecture where as Flume has agent based architecture.
- Data import is event driven in Flume but not is Sqoop.
- In Sqoop data directly goes to HDFS where as in Sqoop data has to flow through channels.

How to get file of equivalent size while importing data using sqoop?

I don't think this question has a particular answer that certainly gives us the required result. Because data is peculiar. You can't decide how it will be. To get this task done we must have complete knowledge how the data is. Whether it has any primary key, size of the data, what kind of column it has and how many of them contains nulls or invalid values.

The simplest answer is if the data don't have a primary key column then it is better to with the argument **--direct-split-size.** Below is the complete syntax of the argument.

--direct-split-size 570000000

This line you can include in your sqoop import command. This will divide the entire imported data into files of size 570 MB. This is one of the ways to get equal sized or equivalent sized files.

Another way is to use --split-by argument. To use this we must have knowledge on one the columns which can be use with this argument. The column must be distributed evenly between smallest and largest values in the column.

KAFKA

https://developer.hpe.com/blog/performance-tuning-of-an-apache-kafkaspark-streaming-system/

https://www.youtube.com/watch?v=bcPMRsI5kT4

https://www.youtube.com/watch?v=kDx8hZhvCQ0

https://www.youtube.com/watch?v=Ai4n_NcKLZQ

https://www.youtube.com/watch?v=kZT8v2_b2XE

https://www.youtube.com/watch?v=Poegm0fF1bo

Data WareHousing and Modelling

data lake vs data warehouse

https://www.talend.com/resources/data-lake-vs-data-warehouse/#:~:text=Data%20lakes%20and%20data%20warehouses,processed%20for%20a%20specific%20purpose.

https://www.analyticsvidhya.com/blog/2020/10/what-is-the-difference-between-data-lake-and-data-warehouse/

rdbms vs dwh vs dl

https://www.mongodb.com/databases/data-lake-vs-data-warehouse-vs-database

dimension table vs fact table

https://stackoverflow.com/questions/20036905/difference-between-fact-table-and-dimension-table

Star schema vs Snowflake schema

https://www.keboola.com/blog/star-schema-vs-snowflake-schema

https://www.datawarehouse4u.info/Data-warehouse-schema-architecture-star-schema.html

OLAP vs OLTP

https://www.datawarehouse4u.info/OLTP-vs-OLAP.html

SCD

https://www.datawarehouse4u.info/SCD-Slowly-Changing-Dimensions.html