```
'General Questions'
-----------------------
1. Tell About Yourself

    I would like to thanks for giving me an opportunity.

    My Name is Vishal Nara. I am from mumbai. I m currently working in ADP Pvt limited from last
     17 Months as Hadoop/Spark developers.
    I have completed in Bachelor in Engineering from Mumbai University.

    My strengths are a positive attitude, quick learning.
    Thats all about me.


2. Rate Yourself in
    Spark 7/10
    Scala 7/10
    Hive  8/10
    Sql   8/10
    Unix  6/10
    Java  5/10

3. Tell About your current Project

    I m currently working in Data Cloud Insights projects.

    We pull the Client Data from Oracle Datawarehouse using Spark Jobs. Generate Cube by
    comparing left and right value
    (For Example Turnover Rate in 2014 vs Turnover Rate in 2015) and generate Difference.

    We can do this for 3 to 4 dimentions like time, job , location.

    From these geneated values we filter most usefull differences and exported to Oracle which
    is used to send mobile notification to Managers.

    Ex : Turnover Rate of your company is 4.0% More than previous year
         Turnover Rate of your company is 2% more in New york location

    Z-Score ?

4. How many years of experience in Spark and Big data Ecosystem

    Initially I started working in Oracle and PL/SQL and In Techmahindra I got opportunity to
    work in Hadoop and Spark.
    And After moving to ADP being in Data Cloud Project I got more opportunity to explore and
    worked in Hive,Sqoop,Spark,Kafka

5. What are roles and Responsibility of you in your team

    I work as Data Engineer in my project. My Roles and Responsibility are

    1. Writing Optimized Oracle Query to get required details from data Warehouse
    2. Data Extraction
    3. Data Cleansing to remove anamolies
    4. Creating Hive Table
    5. Data Processing in Spark using Scala
    6. Data Export to Oracle


6. Explain your Dev and Production cluster

    Dev cluseter :

        1. 15 Node Cluster
        2. 50 TB Hard Disk
        3. 2 TB of RAM
```

```
    Prod Cluster :

        1. 30 Node Cluster
        2. 100 TB Hard Disk
        3. 4 TB of RAM

7. What version that your using for
    Spark
    Hive
    Scala
    Hadoop


'Spark '
--------

1. What is RDDs and why they are immutable

    RDD (Resilient Distributed Dataset) is the fundamental data structure of Apache Spark which
    are an immutable collection of objects which computes on the different node of the cluster.
    Each and every dataset in Spark RDD is logically partitioned across many servers so that
    they can be computed on different nodes of the cluster.

    RDD stands for “Resilient Distributed Dataset”. It is the fundamental data structure of
    Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on
    the different node of the cluster.

    Decomposing the name RDD:

    Resilient, i.e. fault-tolerant with the help of RDD lineage graph(DAG) and so able to
    recompute missing or damaged partitions due to node failures.
    Distributed, since Data resides on multiple nodes.
    Dataset represents records of the data you work with. The user can load the data set
    externally which can be either JSON file, CSV file, text file or database via JDBC with no
    specific data structure.

    There are three ways to create RDDs

    1. Sc.parallelize()
    2. From Other RDD
    3. From Data Sets like csv,json,xmls

2. What is Data Frame

    DataFrame appeared in Spark Release 1.3.0. We can term DataFrame as Dataset organized into
    named columns. DataFrames are similar to the table in a relational database or data frame in
     R /Python. It can be said as a relational table with good optimization technique.

    The idea behind DataFrame is it allows processing of a large amount of structured data.
    DataFrame contains rows with Schema. The schema is the illustration of the structure of data.

    DataFrame in Apache Spark prevails over RDD but contains the features of RDD as well. The
    features common to RDD and DataFrame are immutability, in-memory, resilient, distributed
    computing capability. It allows the user to impose the structure onto a distributed
    collection of data. Thus provides higher level abstraction.

    We can build DataFrame from different data sources. For Example structured data file, tables
     in Hive, external databases or existing RDDs. The Application Programming Interface (APIs)
    of DataFrame is available in various languages. Examples include Scala, Java, Python, and R.

    It makes large data set processing even easier. Data Frame also allows developers to impose
    a structure onto a distributed collection of data. As a result, it allows higher-level
    abstraction.
    Data frame is both space and performance efficient.
    It can deal with both structured and unstructured data formats, for example, Avro, CSV etc .
```

And also storage systems like HDFS, HIVE tables, MySQL, etc.
The DataFrame API's are available in various programming languages. For example Java, Scala, Python, and R.
It provides Hive compatibility. As a result, we can run unmodified Hive queries on existing Hive warehouse.
Catalyst tree transformation uses DataFrame in four phases: a) Analyze logical plan to solve references. b) Logical plan optimization c) Physical planning d) Code generation to compile part of the query to Java bytecode.
It can scale from kilobytes of data on the single laptop to petabytes of data on the large cluster.

3. What is Data Set

Dataset is a data structure in SparkSQL which is strongly typed and is a map to a relational schema. It represents structured queries with encoders. It is an extension to dataframe API . Spark Dataset provides both type safety and object-oriented programming interface. We encounter the release of the dataset in Spark 1.6.

The encoder is primary concept in serialization and deserialization (SerDe) framework in Spark SQL. Encoders translate between JVM objects and Spark's internal binary format. Spark has built-in encoders which are very advanced. They generate bytecode to interact with off-heap data.

An encoder provides on-demand access to individual attributes without having to de-serialize an entire object. To make input output time and space efficient, Spark SQL uses SerDe framework. Since encoder knows the schema of record, it can achieve serialization and deserialization.

Spark Dataset is structured and lazy query expression that triggers on the action. Internally dataset represents logical plan. The logical plan tells the computational query that we need to produce the data. the logical plan is a base catalyst query plan for the logical operator to form a logical query plan. When we analyze this and resolve we can form a physical query plan.

Dataset clubs the features of RDD and DataFrame. It provides:

The convenience of RDD.
Performance optimization of DataFrame.
Static type-safety of Scala.
Thus, Datasets provides a more functional programming interface to work with structured data.

4. Difference between RDDs and Data Frame and Data sets
(https://data-flair.training/blogs/apache-spark-rdd-vs-dataframe-vs-dataset/)

Spark RDD APIs – An RDD stands for Resilient Distributed Datasets. It is Read-only partition collection of records. RDD is the fundamental data structure of Spark. It allows a programmer to perform in-memory computations on large clusters in a fault-tolerant manner. Thus, speed up the task. Follow this link to learn Spark RDD in great detail.

Spark Dataframe APIs – Unlike an RDD, data organized into named columns. For example a table in a relational database. It is an immutable distributed collection of data. DataFrame in Spark allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction. Follow this link to learn Spark DataFrame in detail.

Spark Dataset APIs – Datasets in Apache Spark are an extension of DataFrame API which provides type-safe, object-oriented programming interface. Dataset takes advantage of Spark's Catalyst optimizer by exposing expressions and data fields to a query planner.

'Spark Release'
    RDD – The RDD APIs have been on Spark since the 1.0 release.
    DataFrames – Spark introduced DataFrames in Spark 1.3 release.
    DataSet – Spark introduced Dataset in Spark 1.6 release.

'Data Representation'
    RDD – RDD is a distributed collection of data elements spread across many machines in

```
    the cluster. RDDs are a set of Java or Scala objects representing data.
    DataFrame - A DataFrame is a distributed collection of data organized into named columns
    . It is conceptually equal to a table in a relational database.
    DataSet - It is an extension of DataFrame API that provides the functionality of - type-
    safe, object-oriented programming interface of the RDD API and performance benefits of
    the Catalyst query optimizer and off heap storage mechanism of a DataFrame API.

'Data Formats'
    RDD - It can easily and efficiently process data which is structured as well as
    unstructured. But like Dataframe and DataSets, RDD does not infer the schema of the
    ingested data and requires the user to specify it.
    DataFrame - It can process structured and unstructured data efficiently. It organizes
    the data in the named column. DataFrames allow the Spark to manage schema.
    DataSet - It also efficiently processes structured and unstructured data. It represents
    data in the form of JVM objects of row or a collection of row object. Which is
    represented in tabular forms through encoders.

'Data Sources API'
    RDD - Data source API allows that an RDD could come from any data source e.g. text file,
     a
    database via JDBC etc. and easily handle data with no predefined structure.
    DataFrame - Data source API allows Data processing in different formats (AVRO, CSV, JSON
    , and storage system HDFS, HIVE tables, MySQL). It can read and write from various data
    sources that are mentioned above.
    DataSet - Dataset API of spark also support data from different sources.

'Immutability and Interoperability'
    RDD - RDDs contains the collection of records which are partitioned. The basic unit of
    parallelism in an RDD is called partition. Each partition is one logical division of
    data which is immutable and created through some transformation on existing partitions.
    Immutability helps to achieve consistency in computations. We can move from RDD to
    DataFrame (If RDD is in tabular format) by toDF() method or we can do the reverse by the
     .rdd method. Learn various RDD Transformations and Actions APIs with examples.
    DataFrame - After transforming into DataFrame one cannot regenerate a domain object. For
     example, if you generate testDF from testRDD, then you won’t be able to recover the
    original RDD of the test class.
    DataSet - It overcomes the limitation of DataFrame to regenerate the RDD from Dataframe.
    Datasets allow you to convert your existing RDD and DataFrames into Datasets.

'Compile-time type safety'

    RDD - RDD provides a familiar object-oriented programming style with compile-time type
    safety.
    DataFrame - If you are trying to access the column which does not exist in the table in
    such case Dataframe APIs does not support compile-time error. It detects attribute error
     only at runtime.
    DataSet -  It provides compile-time type safety.
    Learn: Apache Spark vs. Hadoop MapReduce

'Optimization'
    RDD - No inbuilt optimization engine is available in RDD. When working with structured
    data, RDDs cannot take advantages of sparks advance optimizers. For example, catalyst
    optimizer and Tungsten execution engine. Developers optimise each RDD on the basis of
    its attributes.
    DataFrame - Optimization takes place using catalyst optimizer. Dataframes use catalyst
    tree transformation framework in four phases: a) Analyzing a logical plan to resolve
    references. b) Logical plan optimization. c) Physical planning. d) Code generation to
    compile parts of the query to Java bytecode. The brief overview of optimization phase is
     also given in the below figure:
    Spark-SQL-Optimization
    Dataset - It includes the concept of Dataframe Catalyst optimizer for optimizing query
    plan.

'Serialization'
    RDD - Whenever Spark needs to distribute the data within the cluster or write the data
    to disk, it does so use Java serialization. The overhead of serializing individual Java
```

```
    and Scala objects is expensive and requires sending both data and structure between
    nodes.
    DataFrame - Spark DataFrame Can serialize the data into off-heap storage (in memory) in
    binary format and then perform many transformations directly on this off heap memory
    because spark understands the schema. There is no need to use java serialization to
    encode the data. It provides a Tungsten physical execution backend which explicitly
    manages memory and dynamically generates bytecode for expression evaluation.
    DataSet - When it comes to serializing data, the Dataset API in Spark has the concept of
     an encoder which handles conversion between JVM objects to tabular representation. It
    stores tabular representation using spark internal Tungsten binary format. Dataset
    allows performing the operation on serialized data and improving memory use. It allows
    on-demand access to individual attribute without desterilizing the entire object.


'Garbage Collection'
    RDD - There is overhead for garbage collection that results from creating and destroying
     individual objects.
    DataFrame - Avoids the garbage collection costs in constructing individual objects for
    each row in the dataset.
    DataSet - There is also no need for the garbage collector to destroy object because
    serialization
    takes place through Tungsten. That uses off heap data serialization.


'Efficiency/Memory use'
    RDD - Efficiency is decreased when serialization is performed individually on a java and
     scala object which takes lots of time.
    DataFrame - Use of off heap memory for serialization reduces the overhead. It generates
    byte code dynamically so that many operations can be performed on that serialized data.
    No need for deserialization for small operations.
    DataSet - It allows performing an operation on serialized data and improving memory use.
     Thus it allows on-demand access to individual attribute without deserializing the
    entire object.

'Lazy Evolution'
    RDD - Spark evaluates RDDs lazily. They do not compute their result right away. Instead,
     they just remember the transformation applied to some base data set. Spark compute
    Transformations only when an action needs a result to sent to the driver program. Refer
    this guide if you are new to the Lazy Evaluation feature of Spark.
    Apache Spark Lazy Evaluation Feature.

    DataFrame - Spark evaluates DataFrame lazily, that means computation happens only when
    action appears (like display result, save output).
    DataSet - It also evaluates lazily as RDD and Dataset.


5. Difference between Spark 1.0 and Spark 2.0

6. Difference Between Repartitions and coalsec

    1. Both coalesec and repartition enables the re assinging of the partitions at run time.
    2. coalesec by defaultly shuffling is false
    3. Re-partitition by defaultly shuffling is true

    we can swith off shuffing in repartition that will behave as coalesec
    we can not switch on shuffing in coalesec

    repartition is not avaialble in apache storm

    coalesec is re-commendable


    Example coalesec :

    val x = sc.parallelize(Array(1,2,3,4,5),3)
    val y = x.coalesce(2,false)
```

```
    println(y.getNumPartitions)

    Example repartition :

    val x = sc.parallelize(Array(1,2,3,4,5),3)
    val y = x.repartition(2)
    println(y.getNumPartitions)

7. Different kinds of Transformation and Different types of Transformation

        'Types Transformation in Spark'
    --------------------------

    1. Map          --> Realtime
    2. FlatMap      --> Realtime
    3. Filter       --> Realtime
    4. join         --> Realtime
    5. groupbykey   --> Realtime
    6. reduceByKey  --> Realtime
    7. aggregateByKey
    8. mapPartition
    9. mapPartitionWithIndex
    10. coalsec     --> Realtime
    11. repartition  --> Realtime
    12. cogroup
    13. union
    14. union all
    15. distinct
    16. sortBy
    17. intersect
    18. cartesian


    Key-value
    1. aggregateByKey
    2. reduceByKey
    3. groupByKey
    4. sortByKey
    5. join
    6. cogroup


8. Different Actions

    count()
    collect()
    take(n)
    top()
    countByValue()
    reduce()
    fold()
    aggregate()
    foreach()



9.  Features of RDD

    In-memory computation
    Lazy Evaluation
    Fault Tolerance
    Immutability
    Persistence
    Partitioning
    Parallel
    Location-Stickiness
```

```
    Coarse-grained Operation
    Typed
    No limitation

10. Performance Tuning in Spark

11. Difference between Persist vs cache

    Spark RDD persistence is an optimization technique in which saves the result of RDD
    evaluation. Using this we save the intermediate result so that we can use it further if
    required. It reduces the computation overhead.

    We can make persisted RDD through cache() and persist() methods. When we use the cache()
    method we can store all the RDD in-memory. We can persist the RDD in memory and use it
    efficiently across parallel operations.

    The difference between cache() and persist() is that using cache() the default storage level
     is MEMORY_ONLY while using persist() we can use various storage levels (described below).
    It is a key tool for an interactive algorithm. Because, when we persist RDD each node stores
     any partition of it that it computes in memory and makes it reusable for future use. This
    process speeds up the further computation ten times.

    There are some advantages of RDD caching and persistence mechanism in spark. It makes the
    whole system

    Time efficient
    Cost efficient
    Lessen the execution time.

    Using persist() we can use various storage levels to Store Persisted RDDs in Apache Spark.
    1. MEMORY_ONLY
    2. MEMORY_AND_DISK
    3. MEMORY_ONLY_SER
    4. MEMORY_AND_DISK_SER
    5. DISK_ONLY
    6. MEMORY_ONLY_2
    7. MEMORY_AND_DISK_2

RDD.unpersist() --> To Unpersisit


12. What is Spark SQL

    Apache Spark SQL is a module for structured data processing in Spark. Using the interface
    provided by Spark SQL we get more information about the structure of the data and the
    computation performed. With this extra information, one can achieve extra optimization in
    Apache Spark. We can interact with Spark SQL in various ways like DataFrame and the Dataset
    API. The Same execution engine is used while computing a result, irrespective of which API/
    language we use to express the computation. Thus, the user can easily switch back and forth
    between different APIs, it provides the most natural way to express a given transformation.

In Apache Spark SQL we can use structured and semi-structured data in three ways:

To simplify working with structured data it provides DataFrame abstraction in Python, Java, and
Scala. DataFrame is a distributed collection of data organized into named columns. It provides a
 good optimization technique.
The data can be read and written in a variety of structured formats. For example, JSON, Hive
Tables, and Parquet.
Using SQL we can query data, both from inside a Spark program and from external tools. The
external tool connects through standard database connectors (JDBC/ODBC) to Spark SQL.
The best way to use Spark SQL is inside a Spark application. This empowers us to load data and
query it with SQL. At the same time, we can also combine it with “regular” program code in
Python, Java or Scala.


    There were some limitations with RDDs. When working with structured data, there was no
```

```
    inbuilt optimization engine. On the basis of attributes, the developer optimized each RDD.
    Also, there was no provision to handle structured data. The DataFrame in Spark SQL overcomes
     these limitations of RDD. Spark DataFrame is Spark 1.3 release. It is a distributed
    collection of data ordered into named columns. Concept wise it is equal to the table in a
    relational database or a data frame in R/Python. We can create DataFrame using:

    Structured data files
    Tables in Hive
    External databases
    Using existing RDD

     Spark SQL Datasets
    Spark Dataset is an interface added in version Spark 1.6. it is a distributed collection of
    data. Dataset provides the benefits of RDDs along with the benefits of Apache Spark SQL's
    optimized execution engine. Here an encoder is a concept that does conversion between JVM
    objects and tabular representation.

    A Dataset can be made using JVM objects and after that, it can be manipulated using
    functional transformations (map, filter etc.). The Dataset API is accessible in Scala and
    Java. Dataset API is not supported by Python. But because of the dynamic nature of Python,
    many benefits of Dataset API are available. The same is the case with R. Using a Dataset of
    rows we represent DataFrame in Scala and Java. Follow this comparison guide to learn the
    comparison between Java vs Scala.

    Spark Catalyst Optimizer
    The optimizer used by Spark SQL is Catalyst optimizer. It optimizes all the queries written
    in Spark SQL and DataFrame DSL. The optimizer helps us to run queries much faster than their
     counter RDD part. This increases the performance of the system.

    Spark Catalyst is a library built as a rule-based system. And each rule focusses on the
    specific optimization. For example, ConstantFolding focus on eliminating constant expression
     from the query.

    Uses of Apache Spark SQL
    It executes SQL queries.
    We can read data from existing Hive installation using SparkSQL.
    When we run SQL within another programming language we will get the result as Dataset/
    DataFrame

Advantages of Spark SQL
    1. Integrated
    2. Unified Data Access
    3. High compatibility
    4. Standard Connectivity
    5.  Performance Optimization
    6. For batch processing of Hive tables

Disadvantages :
    a. Unsupportive Union type
    b. No error for oversize of varchar type
    c. No support for transactional table
    d. Unsupportive Char type
    e. No support for time-stamp in Avro table.


13. How Fault tolerant achieved in Spark

    The basic fault-tolerant semantic of Spark are:
    Since all RDD is an immutable data set. Each RDD keeps track of the lineage of the
    deterministic operation that employee on fault-tolerant input dataset to create it.

    If any partition of an RDD is lost due to a worker node failure, then that partition can be
    re-computed from the original fault-tolerant dataset using the lineage of operations.

    Assuming that all of the RDD transformations are deterministic, the data in the final
    transformed RDD will always be the same irrespective of failures in the Spark cluster.
```

To achieve fault tolerance for all the generated RDDs, the achieved data replicates among multiple Spark executors in worker node in the cluster. This result in two types of data that should recover in the event of failure:
Data received and replicated – In this, the data replicates on one of the other nodes. Thus we can retrieve data when a failure occurs.

Data received but buffered for replication – the data does not replicate. Thus the only way to recover fault is by retrieving it again from the source.
Failure can also occur in worker and driver nodes.
Failure of worker node – The node which runs the application code on the cluster is worker node. These are the slave nodes. Any of the worker nodes running executor can fail, thus resulting in loss of in-memory data. If any receivers were running on failed nodes, then their buffer data will vanish.
Failure of driver node – If the driver node running the Spark Streaming application fails, then there is the loss of SparkContent. All executors along with their in-memory data vanishes.

14. What version you are using in Spark

2.1

15. Code Sample 1.x and 2.x

1.x

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

object Wordcount {
 def main(args: Array[String]) {

 val conf = new SparkConf().setAppName("WordCount")
 val sc = new SparkContext(conf)
 if (args.length < 2) {
 println("Usage: ScalaWordCount <input> <output>")
 System.exit(1)
 }
 val rawData = sc.textFile(args(0))
 val words = rawData.flatMap(line => line.split(" "))
 val wordCount = words.map(word => (word, 1)).reduceByKey(_ + _)
 wordCount.saveAsTextFile(args(1))
 sc.stop
 }
}
```

2.x

```
import org.apache.spark.sql.SparkSession

object WordCount {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder.master("local[*]").appName("word count").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    println("Success")
    val rawData = sc.textFile(
    "C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\wordcount.txt")
    println("Data SUCCESSFULL")
    val words = rawData.flatMap(line => line.split(" "))
    val wordCount = words.map(word => (word, 1)).reduceByKey(_ + _)
    println(wordCount.count())
```

```
        wordCount.foreach(println)
        wordCount.saveAsTextFile(
        "C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\output")
        spark.stop()

      }
    }
```

16. What is Lineage Graph in Spark and how does it helps in fault tolerant

The basic fault-tolerant semantic of Spark are:
Since all RDD is an immutable data set. Each RDD keeps track of the lineage of the deterministic operation that employee on fault-tolerant input dataset to create it.

If any partition of an RDD is lost due to a worker node failure, then that partition can be re-computed from the original fault-tolerant dataset using the lineage of operations.

Assuming that all of the RDD transformations are deterministic, the data in the final transformed RDD will always be the same irrespective of failures in the Spark cluster.
To achieve fault tolerance for all the generated RDDs, the achieved data replicates among multiple Spark executors in worker node in the cluster. This result in two types of data that should recover in the event of failure:
Data received and replicated – In this, the data replicates on one of the other nodes. Thus we can retrieve data when a failure occurs.

Data received but buffered for replication – the data does not replicate. Thus the only way to recover fault is by retrieving it again from the source.
Failure can also occur in worker and driver nodes.
Failure of worker node – The node which runs the application code on the cluster is worker node. These are the slave nodes. Any of the worker nodes running executor can fail, thus resulting in loss of in-memory data. If any receivers were running on failed nodes, then their buffer data will vanish.
Failure of driver node – If the driver node running the Spark Streaming application fails, then there is the loss of SparkContent. All executors along with their in-memory data vanishes.

17. Why Data Set are faster than Data Frame

Along with all the above benefits, you cannot overlook the space efficiency and performance gains in using DataFrames and Dataset APIs for two reasons.

First, because DataFrame and Dataset APIs are built on top of the Spark SQL engine, it uses Catalyst to generate an optimized logical and physical query plan. Across R, Java, Scala, or Python DataFrame/Dataset APIs, all relation type queries undergo the same code optimizer, providing the space and speed efficiency. Whereas the Dataset[T] typed API is optimized for data engineering tasks, the untyped Dataset[Row] (an alias of DataFrame) is even faster and suitable for interactive analysis.

Second, since Spark as a compiler understands your Dataset type JVM object, it maps your type-specific JVM object to Tungsten's internal memory representation using Encoders. As a result, Tungsten Encoders can efficiently serialize/deserialize JVM objects as well as generate compact bytecode that can execute at superior speeds.

When should I use DataFrames or Datasets?
If you want rich semantics, high-level abstractions, and domain specific APIs, use DataFrame or Dataset.
If your processing demands high-level expressions, filters, maps, aggregation, averages, sum, SQL queries, columnar access and use of lambda functions on semi-structured data, use DataFrame or Dataset.
If you want higher degree of type-safety at compile time, want typed JVM objects, take advantage of Catalyst optimization, and benefit from Tungsten's efficient code generation, use Dataset.

If you want unification and simplification of APIs across Spark Libraries, use DataFrame or Dataset.
If you are a R user, use DataFrames.
If you are a Python user, use DataFrames and resort back to RDDs if you need more control.
Note that you can always seamlessly interoperate or convert from DataFrame and/or Dataset to an RDD, by simple method call .rdd. For instance,

```
// select specific fields from the Dataset, apply a predicate
// using the where() method, convert to an RDD, and show first 10
// RDD rows
val deviceEventsDS = ds.select($"device_name", $"cca3", $"c02_level").where($"c02_level" >
1300)
// convert to RDDs and take the first 10 rows
val eventsRDD = deviceEventsDS.rdd.take(10)
```

Bringing It All Together
In summation, the choice of when to use RDD or DataFrame and/or Dataset seems obvious. While the former offers you low-level functionality and control, the latter allows custom view and structure, offers high-level and domain specific operations, saves space, and executes at superior speeds.

As we examined the lessons we learned from early releases of Spark—how to simplify Spark for developers, how to optimize and make it performant—we decided to elevate the low-level RDD APIs to a high-level abstraction as DataFrame and Dataset and to build this unified data abstraction across libraries atop Catalyst optimizer and Tungsten.

Pick one—DataFrames and/or Dataset or RDDs APIs—that meets your needs and use-case, but I would not be surprised if you fall into the camp of most developers who work with structure and semi-structured data.

18. Role of Encoder and working or Encoder

Project Tungsten will be the largest change to Spark's execution engine since the project's inception. It focuses on substantially improving the efficiency of memory and CPU for Spark applications, to push performance closer to the limits of modern hardware. This effort includes three initiatives:

Memory Management and Binary Processing: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
Cache-aware computation: algorithms and data structures to exploit memory hierarchy
Code generation: using code generation to exploit modern compilers and CPUs
The focus on CPU efficiency is motivated by the fact that Spark workloads are increasingly bottlenecked by CPU and memory use rather than IO and network communication. This trend is shown by recent research on the performance of big data workloads (Ousterhout et al) and we've arrived at similar findings as part of our ongoing tuning and optimization efforts for Databricks Cloud customers.

Why is CPU the new bottleneck? There are many reasons for this. One is that hardware configurations offer increasingly large aggregate IO bandwidth, such as 10Gbps links in networks and high bandwidth SSD's or striped HDD arrays for storage. From a software perspective, Spark's optimizer now allows many workloads to avoid significant disk IO by pruning input data that is not needed in a given job. In Spark's shuffle subsystem, serialization and hashing (which are CPU bound) have been shown to be key bottlenecks, rather than raw network throughput of underlying hardware. All these trends mean that Spark today is often constrained by CPU efficiency and memory pressure rather than IO.

19. How Spark is Better than Hadoop

Apache Spark is lightening fast cluster computing tool. It is up to 100 times faster than Hadoop MapReduce due to its very fast in-memory data analytics processing power.
Apache Spark is a Big Data Framework. Apache Spark is a general purpose data processing engine and is generally used on top of HDFS. Apache Spark is suitable for the variety of data processing requirements ranging from Batch Processing to Data Streaming.
Hadoop is an open source framework which processes data stored in HDFS. Hadoop can process structured, unstructured or semi-structured data. Hadoop MapReduce can process the data only in

Batch mode.
Apache Spark surpasses Hadoop in many cases such as
1. Processing the data in memory which is not possible in Hadoop
2. Processing the data that is in batch, iterative, interactive & streaming i.e. Real Time mode. Whereas Hadoop processes only in batch mode.
3. Spark is faster because it reduces the number of disk read-write operations due to its virtue of storing intermediate data in memory. Whereas in Hadoop MapReduce intermediate output which is output of Map() is always written on local hard disk
4. Apache Spark is easy to program as it has hundreds of high-level operators with RDD (Resilient Distributed Dataset)
5. Apache Spark code is compact due compared to Hadoop MapReduce. Use of Scala makes it very short, reduces programming efforts. Also, Spark provides rich APIs in various languages such as Java, Scala, Python, and R.
6. Spark & Hadoop are both highly fault-tolerant.
7. Spark application running in Hadoop clusters is up to 10 times faster on disk than Hadoop MapReduce.

20. Explain Spark Architecture and Spark Ecosystem

Spark Core - Spark Core is the foundation of the whole project. All the functionality that is in Spark, is present on the top of Spark Core.

Spark Streaming - It allows fault-tolerant streaming of live data streams. It is an add-on to core Spark API. Here it makes use of micro-batching for real-time streaming. Thus it packages live data into small batches and delivers to the batch system for processing.

Spark SQL - Spark SQL component is distributed framework for structured data processing. Using Spark SQL Spark gets more information about the structure of data and the computation being performed. As a result, by using this information Spark can perform extra optimization.

Spark MLlib - MLlib is a scalable learning library that discusses both: High-quality algorithm, High speed. The motive behind MLlib creation is to make machine learning scalable and easy. Thus . it contains machine learning libraries that have an implementation of various machine learning algorithms.

Spark GraphX - GraphX is API for graphs and graph parallel execution. In order to support graph computation, graphX contains set of fundamental operators like sub graph, joinvertices and an optimized variant of Pregel API. Also, clustering, classification, traversal, searching, and pathfinding is possible in graphX.

SparkR - SparkR is Apache Spark 1.4 release. The key component of SparkR is SparkR DataFrame. Data frames are a fundamental data structure for data processing in R and the concept of data frames extends to other languages with libraries like Pandas etc.

21. What is Main Abstraction of Spark

whenever the term basic abstraction in Apache Spark arises, the only name strikes in mind is .. RDD.., RDD stands for “Resilient Distributed Dataset”. It is the fundamental abstraction in Apache Spark. It is the basic data structure. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

RDD stands for “Resilient Distributed Dataset”. It is the fundamental data structure of Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

Resilient, i.e. fault-tolerant with the help of RDD lineage graph(DAG) and so able to recompute missing or damaged partitions due to node failures.
Distributed, since Data resides on multiple nodes.
Dataset represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure

22. How to Integrate Hive and Spark ? And What are its advantages

```
    Spark SQL supports Apache Hive using HiveContext. It uses the Spark SQL execution engine to
    work with data stored in Hive.
    HiveContext is a specialized SQLContext to work with Hive.
    Import org.apache.spark.sql.hive package to use HiveContext
    log4j.logger.org.apache.spark.sql.hive.HiveContext=DEBUG

    SQLContext.sql (or simply sql) allows you to interact with Hive.+

    You can use show functions to learn about the Hive functions supported through the Hive
    integration.


    'How to enable Hive context in Spark 2.x' --****** V Imp

Method :
enablehivesupport

Warehouse Directory :


cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/


val spark= SparkSession
            .builder()
            .appName("Spark Hive Example")
            .config("spark.sql.warehouse.dir","warehouseLocation")
            .enablehivesupport
            .getOrCreate()

'Why DataFrames are very Powerful'


DataFrame = RDD + Catalyst Optimizer + DAG + in Memory

DataSet = RDDs + Catalyst Optimizer + CPU Caches

CBO = Cost Based Opitimizer

Catalyst Optimzer internally uses CBO


23. Pair RDD and Differenet Transformation

    Paired RDDs are the RDD-containing key-value pair. A key-value pair (KYP) contains two
    linked data item. Here Key is the identifier and Value are the data corresponding to the key
     value.

    Transformations :

    Key-value
    1. aggregateByKey
    2. reduceByKey
    3. groupByKey
    4. sortByKey
    5. join
    6. cogroup



24. lazy Evaluation in Spark and its benefits

    The lazy evaluation known as call-by-need is a strategy that delays the execution until one
    requires a value. The transformation in Spark is lazy in nature. Spark evaluate them lazily.
     When we call some operation in RDD it does not execute immediately; Spark maintains the
```

```
    graph of which operation it demands. We can execute the operation at any instance by calling
     the action on the data. The data does not loads until it is necessary.


Read about Spark Lazy Evaluation in detail.
Q.21) What are the benefits of lazy evaluation?
Using lazy evaluation we can:
Increase the manageability of the program.

Saves computation overhead and increases the speed of the system.

Reduces the time and space complexity.

provides the optimization by reducing the number of queries.


25. Json in Hive and Spark

    JSON In Hive

    create table json_table(str String);
    load data local inpath '' into table json_table;
    select get_json_object(str,'$.ecode') as ecode, get_json_object(str,'$.ename') as ename ,
    get_json_object(str,'$.sal') as salary from json_guru;


    import org.apache.spark.sql.SparkSession

object JsonExample {

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

    val spark = SparkSession.builder.master("local[*]").appName("JsonExample").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    //Converting RDD to Data Frame
    import spark.implicits._
    import spark.sql
    val df = sqlContext.read.json("C:\\Users\\NaraVish\\Desktop\\#Personal\\#Interview
    Documents\\filformatsinspark\\world_bank.json")
    df.printSchema() //printing schema
   df.createOrReplaceTempView("jsondata_one")
    df.show()

    //val result =   sqlContext.sql("select url,totalamt,abc.* from jsondata_one " + "lateral
    view explode(theme_namecode) as abc")
    val result =   sqlContext.sql("select _id from jsondata_one")
    result.show(10)

//    println(result)


    //result.write.format("com.databricks.spark.csv").option("header","true").save(
    "C:\\Users\\sonirai\\Desktop\\Hadoop GV\\Spark\\SparkSQL\\datasets\\jsontocsv")
    spark.stop()
  }
}

26. Join Example using Spark Core and Spark SQL

    val edata = sc.textFile("file:///home/cloudera/emp.txt")
val ddata = sc.textFile("file:///home/cloudera/dept.txt")

val edata_pair = edata.map{ x =>
val w = x.split(",")
```

```
val eno = w(0).toInt
val ename = w(1)
val sal = w(2).toInt
val gendar = w(3)
val dno = w(4).toInt
(dno,(eno,ename,sal,gendar))
}

val ddata_pair =ddata.map { x=>
val w = x.split(",")
val dno = w(0).toInt
val dname = w(1)
val dloc = w(2)
(dno,(dname,dloc))
}

val edata_pair_join_ddata_pair = edata_pair.join(ddata_pair)

select dno,loc,avg(sal),max(sal),min(sal) from emp e join dept d
where e.dno=d.dno
group by dno,dloc


import org.apache.spark.sql.SQLContext

val sqlContext = new sqlContext(sc)

val emp = sc.textFile("file:///home/cloudera/emp.txt")
val dept = sc.textFile("file:///home/cloudera/dept.txt")

case class Employee (eno:Int,ename:String,sal:Int,gendar:String,dno:Int)
case class Department (dno:Int,dname:String,dloc:String)


val edata = emp.map{ x =>
val w = x.split(",")
val eno = w(0).toInt
val ename=w(1)
val sal = w(2).toInt
val gendar = w(3)
val dno =w(4).toInt
Employee (eno,ename,sal,gendar,dno)
}


val ddata = dept.map{ x =>
val w = x.split(",")
val dno = w(0).toInt
val dname=w(1)
val dloc = w(2)
Department (dno,dname,dloc)
}

--//converting RDD to DataFrame

import sqlContext.implicits._

val edf = edata.toDF
val ddf = ddata.toDF

edf.show()
ddf.show()

edf.registerTempTable("empview")
ddf.registerTempTable("deptview")
```

```
val eresult= sqlContext.sql("select d.dno,d.dloc,avg(sal) as AVG_SAL ,max(sal) MAX_SAL
,min(sal) MIN_SAL,count(*) COUNT_SAL from empview e join deptview d on e.dno=d.dno group by
d.dno,d.dloc")

val eresult= sqlContext.sql("select d.dno,d.dloc from empview e join deptview d on e.dno=d.dno")
```

27. What is Project Tungsten in Spark

Project Tungsten will be the largest change to Spark's execution engine since the project's inception. It focuses on substantially improving the efficiency of memory and CPU for Spark applications, to push performance closer to the limits of modern hardware. This effort includes three initiatives:

Memory Management and Binary Processing: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
Cache-aware computation: algorithms and data structures to exploit memory hierarchy
Code generation: using code generation to exploit modern compilers and CPUs
The focus on CPU efficiency is motivated by the fact that Spark workloads are increasingly bottlenecked by CPU and memory use rather than IO and network communication. This trend is shown by recent research on the performance of big data workloads (Ousterhout et al) and we've arrived at similar findings as part of our ongoing tuning and optimization efforts for Databricks Cloud customers.

Why is CPU the new bottleneck? There are many reasons for this. One is that hardware configurations offer increasingly large aggregate IO bandwidth, such as 10Gbps links in networks and high bandwidth SSD's or striped HDD arrays for storage. From a software perspective, Spark's optimizer now allows many workloads to avoid significant disk IO by pruning input data that is not needed in a given job. In Spark's shuffle subsystem, serialization and hashing (which are CPU bound) have been shown to be key bottlenecks, rather than raw network throughput of underlying hardware. All these trends mean that Spark today is often constrained by CPU efficiency and memory pressure rather than IO.

28. Why we wont use collect() in production code

When a collect operation is issued on a RDD, the dataset is copied to the driver, i.e. the master node. A memory exception will be thrown if the dataset is too large to fit in memory; takeor takeSamplecan be used to retrieve only a capped number of elements instead.

29. Does Spark Requires Hadoop or not ? Explain

Spark is an in-memory distributed computing engine.
Hadoop is a framework for distributed storage (HDFS) and distributed processing (YARN).
Spark can run with or without Hadoop components (HDFS/YARN)

Distributed Storage:

Since Spark does not have its own distributed storage system, it has to depend on one of these storage systems for distributed computing.
S3 – Non-urgent batch jobs. S3 fits very specific use cases when data locality isn't critical.
Cassandra – Perfect for streaming data analysis and an overkill for batch jobs.
HDFS – Great fit for batch jobs without compromising on data locality.
Distributed processing:
You can run Spark in three different modes: Standalone, YARN and Mesos

30. What is Broadcast Variable and Accumulators and What are its usage

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner. Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

Spark actions are executed through a set of stages, separated by distributed "shuffle" operations. Spark automatically broadcasts the common data needed by tasks within each stage. The data broadcasted this way is cached in serialized form and deserialized before running each task. This means that explicitly creating broadcast variables is only useful when tasks across multiple stages need the same data or when caching the data in deserialized form is important.

Broadcast variables are created from a variable v by calling SparkContext.broadcast(v). The broadcast variable is a wrapper around v, and its value can be accessed by calling the value method. The code below shows this:

Accumulators are variables that are only "added" to through an associative and commutative operation and can therefore be efficiently supported in parallel. They can be used to implement counters (as in MapReduce) or sums. Spark natively supports accumulators of numeric types, and programmers can add support for new types.

As a user, you can create named or unnamed accumulators. As seen in the image below, a named accumulator (in this instance counter) will display in the web UI for the stage that modifies that accumulator. Spark displays the value for each accumulator modified by a task in the "Tasks" table.

31. Where you used Apache Spark in your Project

We used Spark to generate Insight Cube to generate all values for all dimentions.
And also NRT Streaming to read kafka topics

32. Explain Catalyst Framework

Spark SQL is one of the newest and most technically involved components of Spark. It powers both SQL queries and the new DataFrame API. At the core of Spark SQL is the Catalyst optimizer, which leverages advanced programming language features (e.g. Scala's pattern matching and quasiquotes) in a novel way to build an extensible query optimizer.

We recently published a paper on Spark SQL that will appear in SIGMOD 2015 (co-authored with Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, and Ali Ghodsi). In this blog post we are republishing a section in the paper that explains the internals of the Catalyst optimizer for broader consumption.

To implement Spark SQL, we designed a new extensible optimizer, Catalyst, based on functional programming constructs in Scala. Catalyst's extensible design had two purposes. First, we wanted to make it easy to add new optimization techniques and features to Spark SQL, especially for the purpose of tackling various problems we were seeing with big data (e.g., semistructured data and advanced analytics). Second, we wanted to enable external developers to extend the optimizer — for example, by adding data source specific rules that can push filtering or aggregation into external storage systems, or support for new data types. Catalyst supports both rule-based and cost-based optimization.

At its core, Catalyst contains a general library for representing trees and applying rules to manipulate them. On top of this framework, we have built libraries specific to relational query processing (e.g., expressions, logical query plans), and several sets of rules that handle different phases of query execution: analysis, logical optimization, physical planning, and code generation to compile parts of queries to Java bytecode. For the latter, we use another Scala feature, quasiquotes, that makes it easy to generate code at runtime from composable expressions. Finally, Catalyst offers several public extension points, including external data sources and user-defined types.

We use Catalyst's general tree transformation framework in four phases, as shown below: (1) analyzing a logical plan to resolve references, (2) logical plan optimization, (3) physical planning, and (4) code generation to compile parts of the query to Java bytecode. In the physical planning phase, Catalyst may generate multiple plans and compare them based on cost

. All other phases are purely rule-based. Each phase uses different types of tree nodes; Catalyst includes libraries of nodes for expressions, data types, and logical and physical operators. We now describe each of these phases.

33. What are advantages of Parquet File format ? Differnce between Avro and Parquet file

Parquet is an open source file format for Hadoop. Parquet stores nested data structures in a flat columnar format compared to a traditional approach where data is stored in row-oriented approach, parquet is more efficient in terms of storage and performance.

There are several advantages to columnar formats:

1)Organizing by column allows for better compression, as data is more homogeneous. The space savings are very noticeable at the scale of a Hadoop cluster.
2)I/O will be reduced as we can efficiently scan only a subset of the columns while reading the data. Better compression also reduces the bandwidth required to read the input.
3)As we store data of the same type in each column, we can use encoding better suited to the modern processors' pipeline by making instruction branching more predictable.

ParQuet vs Avro

Avro is a row-based storage format for Hadoop.

Parquet is a column-based storage format for Hadoop.

If your use case typically scans or retrieves all of the fields in a row in each query, Avro is usually the best choice.

If your dataset has many columns, and your use case typically involves working with a subset of those columns rather than entire records, Parquet is optimized for that kind of work.

34. Why kairo Serialization is better the Default Java Serialization

Data Serialization
Serialization plays an important role in the performance of any distributed application. Formats that are slow to serialize objects into, or consume a large number of bytes, will greatly slow down the computation. Often, this will be the first thing you should tune to optimize a Spark application. Spark aims to strike a balance between convenience (allowing you to work with any Java type in your operations) and performance. It provides two serialization libraries:

Java serialization: By default, Spark serializes objects using Java's ObjectOutputStream framework, and can work with any class you create that implements java.io.Serializable. You can also control the performance of your serialization more closely by extending java.io. Externalizable. Java serialization is flexible but often quite slow, and leads to large serialized formats for many classes.
Kryo serialization: Spark can also use the Kryo library (version 2) to serialize objects more quickly. Kryo is significantly faster and more compact than Java serialization (often as much as 10x), but does not support all Serializable types and requires you to register the classes you'll use in the program in advance for best performance.

35. Checkpointing in Spark

As an Apache Spark application developer, memory management is one of the most essential tasks, but the difference between caching and checkpointing can cause confusion. Both operations are essential in preventing Spark from having to lazily recompute a resilient distributed dataset (RDD) every time it is referenced, but there are also key differences between the two.

Caching computes and materializes an RDD in memory while keeping track of its lineage (dependencies). There are many levels of persistence supported that allow you to make space and compute cost tradeoffs, and specify the behavior of the RDD when it runs out of memory. Since caching remembers an RDD's lineage, Spark can recompute loss partitions in the event of node failures. Lastly, an RDD that is cached lives within the context of the running application, and once the application terminates, cached RDDs are deleted as well.

Checkpointing saves an RDD to a reliable storage system (e.g. HDFS, S3) while forgetting the RDD's lineage completely. Truncating dependencies becomes relevant especially when the RDD's lineage starts getting long. Checkpointing an RDD is similar to how Hadoop stores intermediate computation values to disk, trading off execution latency with ease of recovering from failures. Since an RDD is checkpointed in an external storage system, it can be reused by other applications.

Now the bigger question is how caching and checkpointing interplay. Let's trace through the compute path of an RDD to find out more.

At the core of Spark's engine is the DAGScheduler that breaks down a job (generated by a Spark action) into a DAG of stages. Each of these shuffle or result stages is further broken down into individual tasks that run on a partition of an RDD. An RDD's iterator method is the entry point for a task to access the underlying data partition. We can see from this method that if the storage level is set, indicating that the RDD may be cached, it first attempts to getOrCompute the partition from the block manager. If the block manager does not have the RDD's partition, it falls back to computeOrReadCheckpoint. As you can guess, computeOrReadCheckpoint retrieves checkpointed values if it exists, and if not, only then is the data partition computed.

All that being said, it is up to you to decide which of the two match your use case at different points in your job. It takes longer to read and write a checkpointed RDD simply because it has to be persisted to an external storage system, but Spark worker failures need not result in a recomputation (assuming the data is intact in the external storage system). On the other hand, cached RDD's will not permanently take up storage space, but recomputation is necessary on worker failure. In general, the length of time it takes to do a computation is a good indicator to use one or the other.

36. MLib in your Project ?

Paycode classification

37. Fold Operation in Spark

Fold is a very powerful operation in spark which allows you to calculate many important values in O(n) time. If you are familiar with Scala collection it will be like using fold operation on collection. Even if you not used fold in Scala, this post will make you comfortable in using fold.

Syntax

```
def fold[T](acc:T)((acc,value) => acc)
```

The above is kind of high level view of fold api. It has following three things

T is the data type of RDD
acc is accumulator of type T which will be return value of the fold operation
A function , which will be called for each element in rdd with previous accumulator.
Let's see some examples of fold

Finding max in a given RDD

Let's first build a RDD

```
val sparkContext = new SparkContext("local", "functional")
val employeeData = List(("Jack",1000.0),("Bob",2000.0),("Carl",7000.0))
val employeeRDD = sparkContext.makeRDD(employeeData)
```

Now we want to find an employee, with maximum salary. We can do that using fold.

To use fold we need a start value. The following code defines a dummy employee as starting accumulator.

```
val dummyEmployee = ("dummy",0.0);
```

Now using fold, we can find the employee with maximum salary.

```
    val maxSalaryEmployee = employeeRDD.fold(dummyEmployee)((acc,employee) => {
    if(acc._2 < employee._2) employee else acc})
    println("employee with maximum salary is"+maxSalaryEmployee)
    Fold by key

    In Map/Reduce key plays a role of grouping values. We can use foldByKey operation to
    aggregate values based on keys.

    In this example, employees are grouped by department name. If you want to find the maximum
    salaries in a given department we can use following code.

    val deptEmployees = List(
          ("cs",("jack",1000.0)),
          ("cs",("bron",1200.0)),
          ("phy",("sam",2200.0)),
          ("phy",("ronaldo",500.0))
        )
      val employeeRDD = sparkContext.makeRDD(deptEmployees)

      val maxByDept = employeeRDD.foldByKey(("dummy",0.0))
      ((acc,element)=> if(acc._2 > element._2) acc else element)

      println("maximum salaries in each dept" + maxByDept.collect().toList)


38. How Spark Can you be used for Data Extraction from RDBMS,
    How it is better than Sqoop


/spark-2.1.0-bin-hadoop2.7/bin/pyspark
--jars "/home/jars/ojdbc6.jar"
--master yarn-client
--num-executors 10
--driver-memory 16g
--executor-memory 8g


empDF = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:oracle:thin:username/password@//hostname:portnumber/SID") \
    .option("dbtable", "hr.emp") \
    .option("user", "db_user_name") \
    .option("password", "password") \
    .option("driver", "oracle.jdbc.driver.OracleDriver") \
    .load()

empDF.printSchema()

empDF.show()

The reason Spark is Faster than Sqoop is Spark works with In-Memory.
Sqoop rights the data to Disk that increases I/O Operation.


39. Roles and Responsibility of
    1. Driver
    2. Executor
    3. Worker Node

    Spark Driver - Master Node of a Spark Application

 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. 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.

The driver program that runs on the master node of the spark cluster schedules the job execution and negotiates with the cluster manager.
It translates the RDD’s into the execution graph and splits the graph into multiple stages.
Driver stores the metadata about all the Resilient Distributed Databases and their partitions.
Cockpits of Jobs and Tasks Execution -Driver program converts a user application into smaller execution units known as tasks. Tasks are then executed by the executors i.e. the worker processes which run individual tasks.
Driver exposes the information about the running spark application through a Web UI at port 4040.
Role of Executor in Spark Architecture

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.
Role of Cluster Manager in Spark Architecture

An external service responsible for acquiring resources on the spark cluster and allocating them to a spark job. There are 3 different types of cluster managers 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.

Choosing a cluster manager for any spark application depends on the goals of the application because all cluster managers provide different set of scheduling capabilities. To get started with apache spark, the standalone cluster manager is the easiest one to use when developing a new spark application.

40. Spark Submit Job Command

```
execute spark-submit --executor-cores 8 --num-executors 16 \
    --executor-memory 35g --master yarn \
    --driver-memory 20g \
    --deploy-mode cluster \
    --name {env}-annual_benchmarks \
       --files /app/dsenv-{env}/dsmain-benchmarks/cook/builder/annual_comp_benchmarks.xml \
    --conf spark.yarn.executor.memoryOverhead=12000 \
    --conf spark.core.connection.ack.wait.timeout=200s \
    --conf spark.yarn.driver.memoryOverhead=12000 \
    /app/dsenv-{env}/dscommon-benchmarkstudio/benchmark_builder/benchmark_builder.py -f
    annual_comp_benchmarks.xml  -p {201706}
```

41. Explain Apache Streaming and How it is Achieved

Spark Streaming is an extension of the core Spark API that allows enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis or plain old TCP sockets and be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark’s machine learning algorithms, and graph processing algorithms on data streams.

Internally, it works as follows. Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data. DStreams can be created either from input data stream from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

This guide shows you how to start writing Spark Streaming programs with DStreams. You can write Spark Streaming programs in Scala or Java, both of which are presented in this guide. You will find tabs throughout this guide that let you choose between Scala and Java code snippets.

```
    import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

// Create a local StreamingContext with two working thread and batch interval of 1 second
val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))

val lines = ssc.socketTextStream("localhost", 9999)

val words = lines.flatMap(_.split(" "))

import org.apache.spark.streaming.StreamingContext._
// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.print()
```

42. Explain D-Stream

A Discretized Stream (DStream), it's the fundamental abstraction in Spark Streaming, is a continuous sequence of RDDs of constant kind representing a steady/nonstop stream of information. DStreams may be created from live data like information/data from TCP sockets, Kafka, Flume, etc employing a StreamingContext or it may be generated by working on existing DStreams exploitation functions like map, window, and reduceByKeyAndWindow. Periodically DStream create an RDD which is generated by a parent DStream.

This category contains the fundamental operations offered on all DStreams, like map, filter, and window. additionally, PairDStreamFunctions contains operations offered solely on DStreams of key-value pairs, like groupByKeyAndWindow and be a part of. Through implicit conversions, these operations are offered on any DStream of pairs (e.g., DStream[(Int, Int)]. DStreams internally is characterized by basic properties: - a listing of alternative DStreams depends on - An amount at that the DStream generates an RDD - operate that's want to generate an RDD once on every occasion interval

Discretized Stream may be a sequence of Resilient Distributed Databases that represent a stream of information. DStreams may be created from varied sources like Apache Kafka, HDFS, and Apache Flume

43. What is Speculative Execution in Spark

Speculative Execution of Tasks

Speculative tasks (also speculatable tasks or task strugglers) are tasks that run slower than most (FIXME the setting) of the all tasks in a job.
Speculative execution of tasks is a health-check procedure that checks for tasks to be speculated, i.e. running slower in a stage than the median of all successfully completed tasks in a taskset (FIXME the setting). Such slow tasks will be re-submitted to another worker. It

will not stop the slow tasks, but run a new copy in parallel.
The thread starts as TaskSchedulerImpl starts in clustered deployment modes with spark.speculation enabled. It executes periodically every spark.speculation.interval after the initial spark.speculation.interval passes.
When enabled, you should see the following INFO message in the logs:
INFO TaskSchedulerImpl: Starting speculative execution thread
It works as task-scheduler-speculation daemon thread pool using j.u.c.ScheduledThreadPoolExecutor with core pool size 1.
The job with speculatable tasks should finish while speculative tasks are running, and it will leave these tasks running - no KILL command yet.
It uses checkSpeculatableTasks method that asks rootPool to check for speculatable tasks. If there are any, SchedulerBackend is called for reviveOffers.

44. What are the Machine Learning algorithm is possible in Spark

Moreover, it provides following ML Algorithms:

Basic statistics
Classification and Regression
Clustering
Collaborative filtering

45. Difference between Spark Session and Spark Context

Spark Context:
Prior to Spark 2.0.0 sparkContext was used as a channel to access all spark functionality.
The spark driver program uses spark context to connect to the cluster through a resource manager (YARN orMesos..).
sparkConf is required to create the spark context object, which stores configuration parameter like appName (to identify your spark driver), application, number of core and memory size of executor running on worker node.

In order to use APIs of SQL, HIVE, and Streaming, separate contexts need to be created.

Example:
creating sparkConf :

```
val conf = new SparkConf().setAppName(“RetailDataAnalysis”).setMaster(“spark://master:7077”).set(“spark.executor.memory”, “2g”)
```

creation of sparkContext:

```
val sc = new SparkContext(conf)
```

Spark Session:

SPARK 2.0.0 onwards, SparkSession provides a single point of entry to interact with underlying Spark functionality and
allows programming Spark with DataFrame and Dataset APIs. All the functionality available with sparkContext are also available in sparkSession.

In order to use APIs of SQL, HIVE, and Streaming, no need to create separate contexts as sparkSession includes all the APIs.

Once the SparkSession is instantiated, we can configure Spark’s run-time config properties.

Example:

Creating Spark session:

```
val spark = SparkSession
.builder
.appName("WorldBankIndex")
.getOrCreate()
```

Configuring properties:

```
spark.conf.set("spark.sql.shuffle.partitions", 6)
```

```
spark.conf.set("spark.executor.memory", "2g")

Spark 2.0.0 onwards, it is better to use sparkSession as it provides access to all the spark
Functionalities that sparkContext does. Also, it provides APIs to work on DataFrames and
Datasets.


46. How do you do logging in Spark Job and how to retrieve

    Log4j in Apache Spark
Spark uses log4j as the standard library for its own logging. Everything that happens inside
Spark gets logged to the shell console and to the configured underlying storage. Spark also
provides a template for app writers so we could use the same log4j libraries to add whatever
messages we want to the existing and in place implementation of logging in Spark.

    Example : ?


47. Difference Betwen
    a. SoryByKey vs distributeByKey
    b. Map vs Map Partition
    c. Map Partition vs Map Partition with Index

        map(func) What does it do? Pass each element of the RDD through the supplied function; i
        .e. func

        flatMap(func) "Similar to map, but each input item can be mapped to 0 or more output
        items (so func should return a Seq rather than a single item)."

        mapPartitions(func) Consider mapPartitions a tool for performance optimization. It won't
         do much for you when running examples on your local machine compared to running across
        a cluster. It's the same as map, but works with Spark RDD partitions. Remember the first
         D in RDD is "Distributed" - Resilient Distributed Datasets. Or, put another way, you
        could say it is distributed over partitions. enter image description here

        mapPartitionsWithIndex(func) Similar to mapPartitions, but also provides a function with
         an Int value to indicate the index position of the partition. enter image description
        here


    d. Repartitions vs coalsec

        The repartition algorithm does a full shuffle and creates new partitions with data that
        's distributed evenly. Let's create a DataFrame with the numbers from 1 to 12.

val x = (1 to 12).toList
val numbersDf = x.toDF("number")
numbersDf contains 4 partitions on my machine.

numbersDf.rdd.partitions.size // => 4
Here is how the data is divided on the partitions:

Partition 00000: 1, 2, 3
Partition 00001: 4, 5, 6
Partition 00002: 7, 8, 9
Partition 00003: 10, 11, 12
Let's do a full-shuffle with the repartition method and get this data on two nodes.

val numbersDfR = numbersDf.repartition(2)
Here is how the numbersDfR data is partitioned on my machine:

Partition A: 1, 3, 4, 6, 7, 9, 10, 12
Partition B: 2, 5, 8, 11
The repartition method makes new partitions and evenly distributes the data in the new
partitions (the data distribution is more even for larger data sets).
```

```
Difference between coalesce and repartition

coalesce uses existing partitions to minimize the amount of data that's shuffled.  repartition
creates new partitions and does a full shuffle.  coalesce results in partitions with different
amounts of data (sometimes partitions that have much different sizes) and repartition results in
 roughly equal sized partitions.

Is coalesce or repartition faster?

coalesce may run faster than repartition, but unequal sized partitions are generally slower to
work with than equal sized partitions. You'll usually need to repartition datasets after
filtering a large data set. I've found repartition to be faster overall because Spark is built
to work with equal sized partitions.

1. Both coalesec and repartition enables the re assinging of the partitions at run time.
2. coalesec by defaultly shuffling is false
3. Re-partitition by defaultly shuffling is true

we can swith off shuffing in repartition that will behave as coalesec
we can not switch on shuffing in coalesec

repartition is not avaialble in apache storm

coalesec is re-commendable



48. How to Identify shuffling in spark

    By Looking at DAG graph , if it shows extra stage that means shuffling happened

49. Common Mistake developers make when it comparately

    People often do mistakes in DAG controlling. So in order to avoid such mistakes. We should
    do the following:

    Always try to use reducebykey instead of groupbykey : The ReduceByKey and GroupByKey can
    perform almost similar functions, but GroupByKey contains large data. Hence, try to use
    ReduceByKey to the most. Make sure you stay away from shuffles as much as possible:
    Always try to lower the side of maps as much as possibleTry not to waste more time in
    PartitioningTry not to shuffle moreTry to keep away from Skews as well as partitions
    tooReduce should be lesser than TreeReduce: Always use TreeReduce instead of Reduce, Because
     TreeReduce does much more work in comparison to the Reduce on the executors.


50. Difference between Spark SQL and Hive

Apache Hive:
----------------
Primarily, its database model is Relational DBMS.
It supports an additional database model, i.e. Key-value store
Basically, it supports all Operating Systems with a Java VM.
It has predefined data types. For example, float or date.
It possesses SQL-like DML and DDL statements.
Apache Hive supports JDBC, ODBC, and Thrift.
We can use several programming languages in Hive. For example C++, Java, PHP, and Python.
It uses data sharding method for storing data on different nodes.
There is a selectable replication factor for redundantly storing data on multiple nodes.
Basically, hive supports concurrent manipulation of data.
Basically, it supports for making data persistent.
There are access rights for users, groups as well as roles.
Schema flexibility and evolution.
Also, can portion and bucket, tables in Apache Hive.
As JDBC/ODBC drivers are available in Hive, we can use it.
It does not offer real-time queries and row level updates.
Also provides acceptable latency for interactive data browsing.
```

```
Hive does not support online transaction processing.
In Apache Hive, latency for queries is generally very high.



Spark SQL:

Primarily, its database model is also Relational DBMS
As similar as Hive, it also supports Key-value store as additional database model.
It supports several operating systems. For example Linux OS, X,  and Windows.
As similar to Spark SQL, it also has predefined data types. For Example, float or date.
Like Apache Hive, it also possesses SQL-like DML and DDL statements.
Spark SQL supports only JDBC and ODBC.
We can use several programming languages in Spark SQL.  For example Java, Python, R, and Scala.
This creates difference between SparkSQL and Hive.
It uses spark core for storing data on different nodes.
Basically, for redundantly storing data on multiple nodes, there is a no replication factor in
Spark SQL.
Whereas, spark SQL also supports concurrent manipulation of data.
As same as Hive, Spark SQL also support for making data persistent.
There are no access rights for users.
Basically, it performs SQL queries.
Through Spark SQL, it is possible to read data from existing Hive installation.
We get the result as Dataset/DataFrame if we run Spark SQL with another programming language.
It does not support union type
Although, no provision of error for oversize of varchar type
It does not support transactional table
However, no support for Char type
It does not support time-stamp in Avro table.



51. Explain sliding window operations

    Spark Streaming also provides windowed computations, which allow you to apply
    transformations over a sliding window of data. The following figure illustrates this sliding
     window.

Spark Streaming

As shown in the figure, every time the window slides over a source DStream, the source RDDs that
 fall within the window are combined and operated upon to produce the RDDs of the windowed
DStream. In this specific case, the operation is applied over the last 3 time units of data, and
 slides by 2 time units. This shows that any window operation needs to specify two parameters.

window length - The duration of the window (3 in the figure).
sliding interval - The interval at which the window operation is performed (2 in the figure).
These two parameters must be multiples of the batch interval of the source DStream (1 in the
figure).

Let’s illustrate the window operations with an example. Say, you want to extend the earlier
example by generating word counts over the last 30 seconds of data, every 10 seconds. To do this
, we have to apply the reduceByKey operation on the pairs DStream of (word, 1) pairs over the
last 30 seconds of data. This is done using the operation reduceByKeyAndWindow.

Sliding Window controls transmission of data packets between various computer networks. Spark
Streaming library provides windowed computations where the transformations on RDDs are applied
over a sliding window of data. Whenever the window slides, the RDDs that fall within the
particular window are combined and operated upon to produce new RDDs of the windowed DStream.



52. Why there are no indexes in spark Sql

    Indexing input data

The fundamental reason why indexing over external data sources is not in the Spark scope is that
 Spark is not a data management system but a batch data processing engine. Since it doesn't own
```

the data it is using it cannot reliably monitor changes and as a consequence cannot maintain indices.
If data source supports indexing it can be indirectly utilized by Spark through mechanisms like predicate pushdown.
Indexing Distributed Data Structures:

standard indexing techniques require persistent and well defined data distribution but data in Spark is typically ephemeral and its exact distribution is nondeterministic.
high level data layout achieved by proper partitioning combined with columnar storage and compression can provide very efficient distributed access without an overhead of creating, storing and maintaining indices.This is a common pattern used by different in-memory columnar systems.
That being said some forms of indexed structures do exist in Spark ecosystem. Most notably Databricks provides Data Skipping Index on its platform.

Other projects, like Succinct (mostly inactive today) take different approach and use advanced compression techniques with with random ac'

53. How Memory Handled in Data Sets
54. What is Data Piping

A data pipeline is a software that consolidates data from multiple sources and makes it available to be used strategically.

The data pipeline architecture consists of several layers:-

1) Data Ingestion
2) Data Collector
3) Data Processing
4) Data Storage
5) Data Query
6) Data Visualization

Let’s get into details of each layer & understand how we can build a real-time data pipeline.

55. How Data Security Achieved in Spark

Spark currently supports authentication via a shared secret. Authentication can be configured to be on via the spark.authenticate configuration parameter. This parameter controls whether the Spark communication protocols do authentication using the shared secret . This authentication is a basic handshake to make sure both sides have the same shared secret and are allowed to communicate. If the shared secret is not identical they will not be allowed to communicate. The shared secret is created as follows:

For Spark on YARN deployments, configuring spark.authenticate to true will automatically handle generating and distributing the shared secret. Each application will use a unique shared secret.
For other types of Spark deployments, the Spark parameter spark.authenticate.secret should be configured on each of the nodes. This secret will be used by all the Master/Workers and applications.

56. Explain Kerberos Security

One of the more confusing topics in Hadoop is how authorization and authentication work in the system. The first and most important thing to recognize is the subtle, yet extremely important, differentiation between authorization and authentication, so let’s define these terms first:

Authentication is the process of determining whether someone is who they claim to be.

Authorization is the function of specifying access rights to resources.

In simpler terms, authentication is a way of proving who I am, and authorization is a way of determining what I can do.

Authentication
If Hadoop is configured with all of its defaults, Hadoop doesn't do any authentication of users. This is an important realization to make, because it can have serious implications in a corporate data center. Let's look at an example of this.

Let's say Joe User has access to a Hadoop cluster. The cluster does not have any Hadoop security features enabled, which means that there are no attempts made to verify the identities of users who interact with the cluster. The cluster's superuser is hdfs, and Joe doesn't have the password for the hdfs user on any of the cluster servers. However, Joe happens to have a client machine which has a set of configurations that will allow Joe to access the Hadoop cluster, and Joe is very disgruntled. He runs these commands:

```
sudo useradd hdfs
sudo -u hdfs hadoop fs -rmr /
1
2
sudo useradd hdfs
sudo -u hdfs hadoop fs -rmr /
```

The cluster goes off and does some work, and comes back and says "Ok, hdfs, I deleted everything!".

So what happened here? Well, in an insecure cluster, the NameNode and the JobTracker don't require any authentication. If you make a request, and say you're hdfs or mapred, the NN/JT will both say "ok, I believe that," and allow you to do whatever the hdfs or mapred users have the ability to do.

Hadoop has the ability to require authentication, in the form of Kerberos principals. Kerberos is an authentication protocol which uses "tickets" to allow nodes to identify themselves. If you need a more in depth introduction to Kerberos, I strongly recommend checking out the Wikipedia page.

Hadoop can use the Kerberos protocol to ensure that when someone makes a request, they really are who they say they are. This mechanism is used throughout the cluster. In a secure Hadoop configuration, all of the Hadoop daemons use Kerberos to perform mutual authentication, which means that when two daemons talk to each other, they each make sure that the other daemon is who it says it is. Additionally, this allows the NameNode and JobTracker to ensure that any HDFS or MR requests are being executed with the appropriate authorization level.

Authorization
Authorization is a much different beast than authentication. Authorization tells us what any given user can or cannot do within a Hadoop cluster, after the user has been successfully authenticated. In HDFS this is primarily governed by file permissions.

HDFS file permissions are very similar to BSD file permissions. If you've ever run ls -l in a directory, you've probably seen a record like this:

```
drwxr-xr-x  2 natty hadoop  4096 2012-03-01 11:18 foo
-rw-r--r--  1 natty hadoop    87 2012-02-13 12:48 bar
1
2
drwxr-xr-x  2 natty hadoop  4096 2012-03-01 11:18 foo
-rw-r--r--  1 natty hadoop    87 2012-02-13 12:48 bar
```

On the far left, there is a string of letters. The first letter determines whether a file is a directory or not, and then there are three sets of three letters each. Those sets denote owner, group, and other user permissions, and the "rwx" are read, write, and execute permissions, respectively. The "natty hadoop" portion says that the files are owned by natty, and belong to the group hadoop. As an aside, a stated intention is for HDFS semantics to be "Unix-like when possible." The result is that certain HDFS operations follow BSD semantics, and others are closer to Unix semantics.

The real question here is: what is a user or group in Hadoop? The answer is: they're strings of characters. Nothing more. Hadoop will very happily let you run a command like

```
hadoop fs -chown fake_user:fake_group /test-dir
1
hadoop fs -chown fake_user:fake_group /test-dir
```

The downside to doing this is that if that user and group really don't exist, no one will be able to access that file except the superusers, which, by default, includes hdfs, mapred, and other members of the hadoop supergroup.

In the context of MapReduce, the users and groups are used to determine who is allowed to submit or modify jobs. In MapReduce, jobs are submitted via queues controlled by the scheduler. Administrators can define who is allowed to submit jobs to particular queues via MapReduce ACLs. These ACLs can also be defined on a job-by-job basis. Similar to the HDFS permissions, if the specified users or groups don't exist, the queues will be unusable, except by superusers, who are always authorized to submit or modify jobs.

The next question to ask is: how do the NameNode and JobTracker figure out which groups a user belongs to?

When a user runs a hadoop command, the NameNode or JobTracker gets some information about the user running that command. Most importantly, it knows the username of the user. The daemons then use that username to determine what groups the user belongs to. This is done through the use of a pluggable interface, which has the ability to take a username and map it to a set of groups that the user belongs to. In a default installation, the user-group mapping implementation forks off a subprocess that runs id -Gn [username]. That provides a list of groups like this:

The Hadoop daemons then use this list of groups, along with the username to determine if the user has appropriate permissions to access the file being requested. There are also other implementations that come packaged with Hadoop, including one that allows the system to be configured to get user-group mappings from an LDAP or Active Directory systems. This is useful if the groups necessary for setting up permissions are resident in an LDAP system, but not in Unix on the cluster hosts.

Something to be aware of is that the set of groups that the NameNode and JobTracker are aware of may be different than the set of groups that a user belongs to on a client machine. All authorization is done at the NameNode/JobTracker level, so the users and groups on the DataNodes and TaskTrackers don't affect authorization, although they may be necessary if Kerberos authentication is enabled. Additionally, it is very important that the NameNode and the JobTracker both be aware of the same groups for any given user, or there may be undefined results when executing jobs. If there's ever any doubt of what groups a user belongs to, hadoop dfsgroups and hadoop mrgroups may be used to find out what groups that a user belongs to, according to the NameNode and JobTracker, respectively.

Putting it all together
A proper, safe security protocol for Hadoop may require a combination of authorization and authentication. Admins should look at their security requirements and determine which solutions are right for them, and how much risk they can take on regarding their handling of data. Additionally, if you are going to enable Hadoop's Kerberos features, I strongly recommend looking into Cloudera Manager, which helps make the Kerberos configuration and setup significantly easier than doing it all by hand.

57. How Execution Starts and Ends of Spark

58. MEMORY_ONLY_2 (2 MEANS WHAT)

    2 Means Replication Factor.

59. Dependencies in RDD

    Dependency class is the base (abstract) class to model a dependency relationship between two or more RDDs.
Dependency has a single method rdd to access the RDD that is behind a dependency.

Whenever you apply a transformation (e.g. map, flatMap) to a RDD you build the so-called RDD lineage graph. Dependency-ies represent the edges in a lineage graph.

NarrowDependency and ShuffleDependency are the two top-level subclasses of Dependency abstract class.

60. What is DAGSchedular

RDDs are formed after every transformation. At high level when we apply action on these RDD, Spark creates a DAG. DAG is a finite directed graph with no directed cycles.

There are so many vertices and edges, where each edge is directed from one vertex to another. It contains a sequence of vertices such that every edge is directed from earlier to later in the sequence. It is a strict generalization of MapReduce model. DAG lets you get into the stage and expand in detail on any stage.

In the stage view, the details of all RDDs that belong to that stage are expanded.

The limitations of Hadoop MapReduce became a key point to introduce DAG in Spark. The computation through MapReduce is carried in three steps:

The data is read from HDFS.
Map and Reduce operations are applied.
The computed result is written back to HDFS.

The interpreter is the first layer, using a Scala interpreter, Spark interprets the code with some modifications.
Spark creates an operator graph when you enter your code in Spark console.
When an Action is called on Spark RDD at a high level, Spark submits the operator graph to the DAG Scheduler.
Operators are divided into stages of the task in the DAG Scheduler. A stage contains task based on the partition of the input data. The DAG scheduler pipelines operators together. For example, map operators are scheduled in a single stage.
The stages are passed on to the Task Scheduler. It launches task through cluster manager. The dependencies of stages are unknown to the task scheduler.
The Workers execute the task on the slave.

DD lineage.

How is Fault tolerance achieved through DAG?
 Aapche Spark Interview Questions and Answers
RDD is split into the partition and each node is operating on a partition at any point in time. Here, the series of Scala function is executing on a partition of the RDD. These operations are composed together and Spark execution engine view these as DAG (Directed Acyclic Graph).

When any node crashes in the middle of any operation say O3 which depends on operation O2, which in turn O1. The cluster manager finds out the node is dead and assign another node to continue processing. This node will operate on the particular partition of the RDD and the series of operation that it has to execute (O1->O2->O3). Now there will be no data loss.

Working of DAG Optimizer in Spark
The DAG in Apache Spark is optimized by rearranging and combining operators wherever possible. For, example if we submit a spark job which has a map() operation followed by a filter operation. The DAG Optimizer will rearrange the order of these operators since filtering will reduce the number of records to undergo map operation.

Advantages of DAG in Spark

```
There are multiple advantages of Spark DAG, let's discuss them one by one:

The lost RDD can be recovered using the Directed Acyclic Graph.
Map Reduce has just two queries the map, and reduce but in DAG we have multiple levels. So to
execute SQL query, DAG is more flexible.
DAG helps to achieve fault tolerance. Thus the lost data can be recovered.
It can do a better global optimization than a system like Hadoop MapReduce.

61. What is task with respect to Spark Job Execution

A task is a unit of work that is sent to the executor. Each stage has some task, one task per
partition. The Same task is done over different partitions of RDD.

62. Explain Data Locality with respect to Spark

Spark relies on data locality, aka data placement or proximity to data source, that makes Spark
jobs sensitive to where the data is located. It is therefore important to have Spark running on
Hadoop YARN cluster if the data comes from HDFS.
In Spark on YARN Spark tries to place tasks alongside HDFS blocks.
With HDFS the Spark driver contacts NameNode about the DataNodes (ideally local) containing the
various blocks of a file or directory as well as their locations (represented as InputSplits),
and then schedules the work to the SparkWorkers.
Spark's compute nodes / workers should be running on storage nodes.
Concept of locality-aware scheduling.
Spark tries to execute tasks as close to the data as possible to minimize data transfer (over
the wire).

Figure 1. Locality Level in the Spark UI
There are the following task localities (consult org.apache.spark.scheduler.TaskLocality object):
PROCESS_LOCAL
NODE_LOCAL
NO_PREF
RACK_LOCAL
ANY

Task location can either be a host or a pair of a host and an executor.


63. Why Spark is superior than Hadoop

Cost Efficient - In Hadoop, during replication, a large number of servers, huge amount of
storage, and the large data center is required. Thus, installing and using Apache Hadoop is
expensive. While using Apache Spark is a cost effective solution for big data environment.
Performance - The basic idea behind Spark was to improve the performance of data processing. And
 Spark did this to 10x-100x times. And all the credit of faster processing in Spark goes to in-
memory processing of data. In Hadoop, the data processing takes place in disc while in Spark the
 data processing takes place in memory. It moves to the disc only when needed. The Spark in-
memory computation is beneficial for iterative algorithms. When it comes to performance, because
 of batch processing in Hadoop it's processing is quite slow while the processing speed of
Apache is faster as it supports micro-batching.
Ease of development - The core in Spark is the distributed execution engine. Various languages
are supported by Apache Spark for distributed application development. For example, Java, Scala,
 Python, and R. On the top of spark core, various libraries are built that enables workload.
they make use of streaming, SQL, graph and machine learning. Hadoop also supports some of these
workloads but Spark eases the development by combining all into the same application. d. Failure
 recovery: The method of Fault
Failure recovery - The method of Fault Recovery is different in both Apache Hadoop and Apache
Spark. In Hadoop after every operation data is written to disk. The data objects are stored in
Spark in RDD distributed across data cluster. The RDDs are either in memory or on disk and
provides full recovery from faults or failure.
File Management System - Hadoop has its own File Management System called HDFS (Hadoop
Distributed File System). While Apache Spark an integration with one, it may be even HDFS. Thus,
 Hadoop can run over Apache Spark.
Computation model - Apache Hadoop uses batch processing model i.e. it takes a large amount of
data and processes it. But Apache Spark adopts micro-batching. Must for handling near real time
processing data model. When it comes to performance, because of batch processing in Hadoop it's
```

```
processing is quite slow. The processing speed of Apache is faster as it supports micro-batching.
Lines of code – Apache Hadoop has near about 23, 00,000 lines of code while Apache Spark has 20,
000 lines of code.
Caching – By caching partial result in memory of distributed workers Spark ensures low latency
computations. While MapReduce is completely disk oriented, there is no provision of caching.
Scheduler – Because of in-memory computation in Spark, it acts as its own flow scheduler. While
with Hadoop MapReduce we need an extra job scheduler like Azkaban or Oozie so that we can
schedule complex flows.
Spark API – Because of very Strict API in Hadoop MapReduce, it is not versatile. But since Spark
 discards many low-level details it is more productive.
Window criteria – Apache Spark has time-based window criteria. But Apache Hadoop does not have
window criteria since it does not support streaming.
Faster – Apache Hadoop executes job 10 to 100 times faster than Apache Hadoop MapReduce.
License – Both Apache Hadoop and Apache MapReduce has a License Version 2.0.
DAG() – In Apache Spark, there is cyclic data flow in machine learning algorithm, which is a
direct acyclic graph. While in Hadoop MapReduce data flow does not have any loops, rather it is
a chain of the image.
Memory Management – Apache Spark has automatic memory management system. While Memory Management
 in Apache Hadoop can be either statistic or dynamic.
Iterative Processing – In Apache Spark, the data iterates in batches. Here processing and
scheduling of each iteration are separate. While in Apache Hadoop there is no provision for
iterative processing.
Latency – The time taken for processing by Apache Spark is less as compared to Hadoop since it
caches its data on memory by means of RDD, thus the latency of Apache Spark is less as compared
to Hadoop.



'Scala'
--------

1. Features of Scala

    There are following features of scala:

    Type inference
    Singleton object
    Immutability
    Lazy computation
    Case classes and Pattern matching
    Concurrency control
    String interpolation
    Higher order function
    Traits
    Rich collection set

2. What is closure

    A function whose return value depends on variable(s) declared outside it, is a closure. This
     is much like that in Python.

    val sum=(a:Int,b:Float)=>a+b
    scala> sum(2,3)
    res2: Float = 5.0

    scala> var c=7
c: Int = 7
scala> val sum1=(a:Int,b:Int)=>(a+b)*c

scala> sum1(2,3)
res3: Int = 35

So, while ‘sum’ is trivially closed over itself, ‘sum1’ refers to ‘c’ every time we call it, and
 reads its current value. Let’s try changing the value of c:

3. What is currying
```

Currying is the technique of transforming a function with multiple arguments into a function with just one argument. The single argument is the value of the first argument from the original function and the function returns another single argument function. This in turn would take the second original argument and itself return another single argument function. This chaining continues over the number of arguments of the original. The last in the chain will have access to all of the arguments and so can do whatever it needs to do.

Methods may define multiple parameter lists. When a method is called with a fewer number of parameter lists, then this will yield a function taking the missing parameter lists as its arguments.

Here is an example:

```
object CurryTest extends App {

  def filter(xs: List[Int], p: Int => Boolean): List[Int] =
    if (xs.isEmpty) xs
    else if (p(xs.head)) xs.head :: filter(xs.tail, p)
    else filter(xs.tail, p)

  def modN(n: Int)(x: Int) = ((x % n) == 0)

  val nums = List(1, 2, 3, 4, 5, 6, 7, 8)
  println(filter(nums, modN(2)))
  println(filter(nums, modN(3)))
}
```

Note: method modN is partially applied in the two filter calls; i.e. only its first argument is actually applied. The term modN(2) yields a function of type Int => Boolean and is thus a possible candidate for the second argument of function filter.

Here’s the output of the program above:

```
List(2,4,6,8)
List(3,6)
```

4. Method Overiding and Method overloading

When a subclass has the same name method as defined in the parent class, it is known as method overriding. When subclass wants to provide a specific implementation for the method defined in the parent class, it overrides method from parent class.

Scala Method Overloading

Scala provides method overloading feature which allows us to define methods of same name but having different parameters or data types. It helps to optimize code.

Scala Method Overloading Example by using Different Parameters

In the following example, we have define two add methods with different number of parameters but having same data type.

5. Difference between val and var

A var is a variable. It’s a mutable reference to a value. Since it’s mutable, its value may change through the program lifetime. Keep in mind that the variable type cannot change in Scala. You may say that a var behaves similarly to Java variables.
A val is a value. It’s an immutable reference, meaning that its value never changes. Once assigned it will always keep the same value. It’s similar to constants in another languages.
A def creates a method (and a method is different from a function - thanks to AP for his comment). It is evaluated on call.

6. How Exception can be handled in Scala

Exception handling is a mechanism which is used to handle abnormal conditions. You can also avoid termination of your program unexpectedly.

Scala makes "checked vs unchecked" very simple. It doesnt have checked exceptions. All exceptions are unchecked in Scala, even SQLException and IOException.

```
class ExceptionExample{
    def divide(a:Int, b:Int) = {
            a/b             // Exception occurred here
        println("Rest of the code is executing...")
    }
}
object MainObject{
    def main(args:Array[String]){
        var e = new ExceptionExample()
        e.divide(100,0)

    }
}
```

Scala Try Catch

Scala provides try and catch block to handle exception. The try block is used to enclose suspect code. The catch block is used to handle exception occurred in try block. You can have any number of try catch block in your program according to need.

Scala Try Catch Example

In the following program, we have enclosed our suspect code inside try block. After try block we have used a catch handler to catch exception. If any exception occurs, catch handler will handle it and program will not terminate abnormally.

```
class ExceptionExample{
    def divide(a:Int, b:Int) = {
        try{
            a/b
        }catch{
            case e: ArithmeticException => println(e)
        }
        println("Rest of the code is executing...")
    }
}
object MainObject{
    def main(args:Array[String]){
        var e = new ExceptionExample()
        e.divide(100,0)

    }
}
```

The finally block is used to release resources during exception. Resources may be file, network connection, database connection etc. the finally block executes guaranteed. The following program illustrate the use of finally block.

```
class ExceptionExample{
    def divide(a:Int, b:Int) = {
        try{
            a/b
            var arr = Array(1,2)
            arr(10)
        }catch{
            case e: ArithmeticException => println(e)
```

```
            case ex: Exception =>println(ex)
            case th: Throwable=>println("found a unknown exception"+th)
        }
        finally{
            println("Finaly block always executes")
        }
        println("Rest of the code is executing...")
    }
}


object MainObject{
    def main(args:Array[String]){
        var e = new ExceptionExample()
        e.divide(100,10)

    }
}
```

7. What are different transformation in scala

    a. Map
    b. flatMap
    c. join

8. What is Higher Order Functions

Higher order functions take other functions as parameters or return a function as a result. This is possible because functions are first-class values in Scala. The terminology can get a bit confusing at this point, and we use the phrase “higher order function” for both methods and functions that take functions as parameters or that return a function.

One of the most common examples is the higher-order function map which is available for collections in Scala.

```
val salaries = Seq(20000, 70000, 40000)
val doubleSalary = (x: Int) => x * 2
val newSalaries = salaries.map(doubleSalary) // List(40000, 140000, 80000)
```

doubleSalary is a function which takes a single Int, x, and returns x * 2. In general, the tuple on the left of the arrow => is a parameter list and the value of the expression on the right is what gets returned. On line 3, the function doubleSalary gets applied to each element in the list of salaries.

To shrink the code, we could make the function anonymous and pass it directly as an argument to map:

```
val salaries = Seq(20000, 70000, 40000)
val newSalaries = salaries.map(x => x * 2) // List(40000, 140000, 80000)
```

Notice how x is not declared as an Int in the above example. That’s because the compiler can infer the type based on the type of function map expects. An even more idiomatic way to write the same piece of code would be:

```
val salaries = Seq(20000, 70000, 40000)
val newSalaries = salaries.map(_ * 2)
```

Since the Scala compiler already knows the type of the parameters (a single Int), you just need to provide the right side of the function. The only caveat is that you need to use _ in place of a parameter name (it was x in the previous example).

Coercing methods into functions

It is also possible to pass methods as arguments to higher-order functions because the Scala compiler will coerce the method into a function.

```
case class WeeklyWeatherForecast(temperatures: Seq[Double]) {

  private def convertCtoF(temp: Double) = temp * 1.8 + 32

  def forecastInFahrenheit: Seq[Double] = temperatures.map(convertCtoF) // <-- passing the
  method convertCtoF
}
```

Here the method convertCtoF is passed to forecastInFahrenheit. This is possible because the compiler coerces convertCtoF to the function x => convertCtoF(x) (note: x will be a generated name which is guaranteed to be unique within its scope).

Functions that accept functions

One reason to use higher-order functions is to reduce redundant code. Let’s say you wanted some methods that could raise someone’s salaries by various factors. Without creating a higher-order function, it might look something like this:

```
object SalaryRaiser {

  def smallPromotion(salaries: List[Double]): List[Double] =
    salaries.map(salary => salary * 1.1)

  def greatPromotion(salaries: List[Double]): List[Double] =
    salaries.map(salary => salary * math.log(salary))

  def hugePromotion(salaries: List[Double]): List[Double] =
    salaries.map(salary => salary * salary)
}
```

Notice how each of the three methods vary only by the multiplication factor. To simplify, you can extract the repeated code into a higher-order function like so:

```
object SalaryRaiser {

  private def promotion(salaries: List[Double], promotionFunction: Double => Double): List[
  Double] =
    salaries.map(promotionFunction)

  def smallPromotion(salaries: List[Double]): List[Double] =
    promotion(salaries, salary => salary * 1.1)

  def bigPromotion(salaries: List[Double]): List[Double] =
    promotion(salaries, salary => salary * math.log(salary))

  def hugePromotion(salaries: List[Double]): List[Double] =
    promotion(salaries, salary => salary * salary)
}
```

The new method, promotion, takes the salaries plus a function of type Double => Double (i.e. a function that takes a Double and returns a Double) and returns the product.

Functions that return functions

There are certain cases where you want to generate a function. Here’s an example of a method that returns a function.

```
def urlBuilder(ssl: Boolean, domainName: String): (String, String) => String = {
  val schema = if (ssl) "https://" else "http://"
  (endpoint: String, query: String) => s"$schema$domainName/$endpoint?$query"
}

val domainName = "www.example.com"
def getURL = urlBuilder(ssl=true, domainName)
val endpoint = "users"
val query = "id=1"
```

```
val url = getURL(endpoint, query) // "https://www.example.com/users?id=1": String
Notice the return type of urlBuilder (String, String) => String. This means that the returned
anonymous function takes two Strings and returns a String. In this case, the returned anonymous
function is (endpoint: String, query: String) => s"https://www.example.com/$endpoint?$query".

9. What do you mean by First class Functions


down vote
Being "first-class" is not a formally defined notion, but it generally means that an entity has
three properties:

It can be used, without restriction, wherever "ordinary" values can, i.e., passed and returned
from functions, put in containers, etc.
It can be constructed, without restriction, wherever "ordinary" values can, i.e., locally, in an
 expression, etc.
It can be typed in a way similar to "ordinary" values, i.e., there is a type assigned to such an
 entity, and it can be freely composed with other types.
For functions, (2) particularly implies that a local function can use all names in scope, i.e.
you have lexical closures. It also often comes with an anonymous form for construction (such as
anonymous functions), but that is not strictly required (e.g. if the language has general enough
 let-expressions). Point (3) is trivially true in untyped languages.

So you see why functions in Scala (and in functional languages) are called first-class. Here are
 some other examples.

Functions in C/C++ are not first-class. While (1) and (3) are arguably available through
function pointers, (2) is not supported for functions proper. (A point that's often overlooked.)
Likewise, arrays and structs are not first-class in C land.
Classes in Scala are not first-class. You can define and nest them, but not e.g. pass them to a
function (only its instances). There are OO-languages with first-class classes, and in fact,
the so-called nuObj calculus that informed Scala's design also allows that.
First-class modules are an often desired feature in ML-like languages. They are difficult,
because they lead to undecidable type-checking. Some ML dialect allow modules to be wrapped up
as first-class values, but arguably, that does not make modules first-class themselves.

10. How to process XMLs in Scala

11. Advantages of Scala over other Languages


a)      The arrays uses regular generics, while in other language, generics are bolted on as an
afterthought and are completely separate but have overlapping behaviours with arrays.


b)      Scala has immutable “val” as a first class language feature. The “val” of scala is
similar to Java final variables.  Contents may mutate but top  reference is immutable.
c)       Scala lets ‘if blocks’, ‘for-yield loops’, and ‘code’ in braces to return a value. It
is more preferable, and eliminates the need for a separate ternary operator.
d)      Singleton has singleton objects rather than C++/Java/ C# classic static.  It is a
cleaner solution
e)       Persistent immutable collections are the default and built into the standard library.
f)       It has native tuples and a concise code
g)      It has no boiler plate code


12. What is differenec between concurrency and parallilism

People often confuse with the terms concurrency and parallelism. When several computations
execute sequentially during overlapping time periods it is referred to as concurrency whereas
when processes are executed simultaneously it is known as parallelism. Parallel collection,
Futures and Async library are examples of achieving parallelism in Scala.


13. What is Difference between Nil,Null,None,Nothing
```

• Null – It's a sub-type of AnyRef type in Scala Types hierarchy. As Scala runs on JVM, it uses NULL to provide the compatibility with Java null keyword, or in Scala terms, to provide type for null keyword, Null type exists. It represents the absence of type information for complex types that are inherited from AnyRef.
• Nothing – It's a sub-type of all the types exists in Scala Types hierarchy. It helps in providing the return type for the operations that can affect a normal program's flow. It can only be used as a type, as instantiation of nothing cannot be done. It incorporates all types under AnyRef and AnyVal. Nothing is usually used as a return type for methods that have abnormal termination and result in an exception.
• Nil – It's a handy way of initializing an empty list since, Nil, is an object, which extends List [Nothing].
• None – In programming, there are many circumstances, where we unexpectedly received null for the methods we call. In java these are handled using try/catch or left unattended causing errors in the program. Scala provides a very graceful way of handling those situations. In cases, where you don't know, if you would be able to return a value as expected, we can use Option [T]. It is an abstract class, with just two sub-classes, Some [T] and none. With this, we can tell users that, the method might return a T of type Some [T] or it might return none.

14. Explain Data types in Scala

Data Types
a. Scala Byte
b. Scala Short
c. Scala Int
d. Scala Long
e. Scala Float
f. Scala Double
g. Scala Char
h. Scala String
i. Scala Boolean
j. Scala Unit
k. Scala Null
l. Scala Nothing
m. Scala Any
n. Scala AnyVal
o. Scala AnyRef

15. Explain
a. Singleton Object

In Scala, an object is a class with exactly one instance. Like a lazy val, it creates lazily when we reference it. It is a value, and as a top-level value, it is a Scala singleton. To define an object, we use the keyword 'object':

```
scala> object Box
defined object Box
```

The methods we declare inside Scala singleton object are globally accessible, we don't need an object for this. We can import them from anywhere in the program. And since there is no idea of 'static' in Scala, we must provide a point of entry for the program to execute. Scala singleton object makes for this. Without such an object, the code compiles but produces no output.

b. class

c. traits

Traits in Scala are like partially implemented interfaces. It may contain abstract and non-abstract methods. It may be that all methods are abstract, but it should have at least one abstract method. Not only are they similar to Java interfaces, Scala compiles them into those with corresponding implementation classes holding any methods implemented in the traits.

You can say that using Scala trait, we can share interfaces and fields between classes. Within Scala trait, we can declare variables and values. When we do not initialize them, they are abstract. In other cases, the implementing class for the trait internally implements them.

16. Recursion problem in scala
17. What do you understand by 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.

To define a minimal Scala Case Class, we need the keywords 'case class', an identifier, and a parameter list. We can keep the parameter list empty.

So, let's define a class 'Song'.

```
scala> case class Song(title:String,artist:String,track:Int)
```

Creating a Scala Object
And now, it's time to create a Scala Object for this Scala class.

```
scala> val stay=Song("Stay","Inna",4)
stay: Song = Song(Stay,Inna,4)
```

18. Advantages of Having immutability in scala

Scala uses immutability by default in most of the cases as it helps resolve issues when dealing with concurrent programs and any other equality issues.

19. Why Scala preferred than python

type safety . Scala provided compile time Error.

20. Explain scala collection

Scala set is a collection of pairwise elements of the same type. Scala set does not contain any duplicate elements. There are two kinds of sets, mutable and immutable.

Scala map is a collection of key or value pairs. Based on its key any value can be retrieved. Values are not unique but keys are unique in the Map.

21. Explain Object Main Extends App means
22. what is unit in scala

The 'Unit' is a type similar to void in Java. You can say it is a Scala equivalent of the void in Java, while still providing the language with an abstraction over the Java platform. The empty tuple '()' is a term representing a Unit value in Scala.

23. Program to Explain
a. If Else

```
{
val x=17
if(x<10)
{
    println("Woohoo!")
}
else
{
    println("Oh no!")
```

```
    }
    }


    b. For Loop

        object Main extends App {
    var a=7
    for(a<-1 to 10)
    {
        println(a)
    }
    }

    c. case statement

    val gendar = "m"
      val result = gendar match{
      case "m" => "Male"
      case "f" => "Female"
      case other => "Unknown"
      }

24. How does yield work

    yield generates a value to be kept in each iteration of a loop. yield is used in for
    comprehensions as to provide a syntactic alternative to the combined usage of map/flatMap
    and filter operations on monads


25. Explain fold left and fold right

    One of the functional programming tricks in Scala that I recently learned and enjoyed is
    folding, namely the fold, foldLeft and foldRight functions. As implied by their names, the
    three methods share many concepts in common, but there are also subtle differences in their
    implementations.

As I am a firm believer of learning by examples, I put together some code snippets (many thanks
to this post) that hopefully could help you better understand the nitty-gritty of fold, foldLeft
 and foldRight.

Common folding concepts

Folding is a very powerful operation on Scala Collections. First thing first, let's take a look
at the signatures of the three implementations of folding, i.e. fold, foldLeft and foldRight.

def fold[A1 >: A](z: A1)(op: (A1, A1) ⇒ A1): A1
def foldLeft[B](z: B)(op: (B, A) ⇒ B): B
def foldRight[B](z: B)(op: (A, B) ⇒ B): B
In essence, these functions process a data structure recursively through use of a pre-defined
combining operation op and an initial value z, then gives a return value. If used correctly,
these methods can often do a lot of complicated things with a small amount of code.

To illustrate the common concepts among the three folding functions (we will save the explantion
 of their differences for the next section), I will take foldLeft as an example here for 1) it
is relatively easier to understand and 2) it arguably is the most frequently used folding
technique at least based on my experiences.

26. How do you handle regular expression in scala

Regular expressions are strings which can be used to find patterns (or lack thereof) in data.
Any string can be converted to a regular expression using the .r method.

import scala.util.matching.Regex

val numberPattern: Regex = "[0-9]".r
```

```
numberPattern.findFirstMatchIn("awesomepassword") match {
  case Some(_) => println("Password OK")
  case None => println("Password must contain a number")
}
```

In the above example, the numberPattern is a Regex (regular expression) which we use to make sure a password contains a number.

You can also search for groups of regular expressions using parentheses.

```
import scala.util.matching.Regex

val keyValPattern: Regex = "([0-9a-zA-Z-#() ]+): ([0-9a-zA-Z-#() ]+)".r

val input: String =
  """background-color: #A03300;
    |background-image: url(img/header100.png);
    |background-position: top center;
    |background-repeat: repeat-x;
    |background-size: 2160px 108px;
    |margin: 0;
    |height: 108px;
    |width: 100%;""".stripMargin

for (patternMatch <- keyValPattern.findAllMatchIn(input))
  println(s"key: ${patternMatch.group(1)} value: ${patternMatch.group(2)}")
```

Here we parse out the keys and values of a String. Each match has a group of sub-matches. Here is the output:

```
key: background-color value: #A03300
key: background-image value: url(img
key: background-position value: top center
key: background-repeat value: repeat-x
key: background-size value: 2160px 108px
key: margin value: 0
key: height value: 108px
key: width value: 100
```

27. Explain Annotations

Scala Annotations let us associate meta-information with definitions. We apply an annotation clause to the first definition or the declaration following it. We can use multiple annotations before a definition or declaration in any order.

Scala annotation is of the form @c or @c(a1,…,an), where c is a constructor for class C, which conforms to the scala.Annotation class.

One such annotation is @deprecated. When we put this before a method, the compiler issues a warning when we use this method. Let's take an example of Scala Annotations.

Let's Study Scala Method Overloading with Example

```
scala> @deprecated
| def sayhello()={"hello"}
<console>:11: warning: @deprecated now takes two arguments; see the scaladoc.
@deprecated
^
sayhello: ()String
scala> print(sayhello())
<console>:13: warning: method sayhello is deprecated
print(sayhello())
^
Hello
```

This lets us use the method; it returns a warning, but not an error.

We can attach an annotation to a variable, an expression, a method, or any other element. This can be a class, an object, a trait, or anything else. When with a declaration or a definition, it appears in front; when with a type, it appears after. With an expression, it appears after and is separated by a colon. To an entity, we can apply more than one such annotation. Here's an example:

For classes: @deprecated("Use D", "1.0") class C { … }

For types: String @local

For variables: @transient @volatile var m: Int

For expressions: (e: @unchecked) match { … }

31. Explain Singleton and Companion objects

Singleton Objects
In Scala, an object is a class with exactly one instance. Like a lazy val, it creates lazily when we reference it. It is a value, and as a top-level value, it is a Scala singleton. To define an object, we use the keyword 'object':

```
scala> object Box
defined object Box
```

The methods we declare inside Scala singleton object are globally accessible, we don't need an object for this. We can import them from anywhere in the program. And since there is no idea of 'static' in Scala, we must provide a point of entry for the program to execute. Scala singleton object makes for this. Without such an object, the code compiles but produces no output.

Scala Companion Object
Coming from Scala singleton objects, we now discuss companion objects. A Scala companion object is an object with the same name as a class. We can call it the object's companion class. The companion class-object pair is to be in a single source file. Either member of the pair can access its companion's private members. Let's take an example.

```
scala> class CompanionClass{
| def greet(){
| println("Hello")
| }
| }
defined class CompanionClass
scala> object CompanionObject{
| def main(args:Array[String]){
| new CompanionClass().greet()
| println("Companion object")
| }
| }
defined object CompanionObject
```

So, this was all about Scala Object Tutorial

32. Explain String Interpolation

Introduction

Starting in Scala 2.10.0, Scala offers a new mechanism to create strings from your data: String Interpolation. String Interpolation allows users to embed variable references directly in processed string literals. Here's an example:

```
val name = "James"
println(s"Hello, $name")  // Hello, James
```

In the above, the literal s"Hello, $name" is a processed string literal. This means that the compiler does some additional work to this literal. A processed string literal is denoted by a set of characters preceding the ". String interpolation was introduced by SIP-11, which contains all details of the implementation.

Usage

Scala provides three string interpolation methods out of the box: s, f and raw.

The s String Interpolator

Prepending s to any string literal allows the usage of variables directly in the string. You've already seen an example here:

```
val name = "James"
println(s"Hello, $name")  // Hello, James
```

Here $name is nested inside an s processed string. The s interpolator knows to insert the value of the name variable at this location in the string, resulting in the string Hello, James. With the s interpolator, any name that is in scope can be used within a string.

String interpolators can also take arbitrary expressions. For example:

```
println(s"1 + 1 = ${1 + 1}")
```

will print the string 1 + 1 = 2. Any arbitrary expression can be embedded in ${}.

The f Interpolator

Prepending f to any string literal allows the creation of simple formatted strings, similar to printf in other languages. When using the f interpolator, all variable references should be followed by a printf-style format string, like %d. Let's look at an example:

```
val height = 1.9d
val name = "James"
println(f"$name%s is $height%2.2f meters tall")  // James is 1.90 meters tall
```

The f interpolator is typesafe. If you try to pass a format string that only works for integers but pass a double, the compiler will issue an error. For example:

```
val height: Double = 1.9d

scala> f"$height%4d"
<console>:9: error: type mismatch;
 found   : Double
 required: Int
           f"$height%4d"
              ^
```

The f interpolator makes use of the string format utilities available from Java. The formats allowed after the % character are outlined in the Formatter javadoc. If there is no % character after a variable definition a formatter of %s (String) is assumed.

The raw Interpolator

The raw interpolator is similar to the s interpolator except that it performs no escaping of literals within the string. Here's an example processed string:

```
scala> s"a\nb"
res0: String =
a
b
```

Here the s string interpolator replaced the characters \n with a return character. The raw interpolator will not do that.

```
scala> raw"a\nb"
res1: String = a\nb
```

The raw interpolator is useful when you want to avoid having expressions like \n turn into a return character.

In addition to the three default string interpolators, users can define their own.

Advanced Usage

In Scala, all processed string literals are simple code transformations. Anytime the compiler encounters a string literal of the form:

```
id"string content"
```

it transforms it into a method call (id) on an instance of StringContext. This method can also be available on implicit scope. To define our own string interpolation, we simply need to create an implicit class that adds a new method to StringContext. Here's an example:

```
// Note: We extends AnyVal to prevent runtime instantiation.  See
// value class guide for more info.
implicit class JsonHelper(val sc: StringContext) extends AnyVal {
  def json(args: Any*): JSONObject = sys.error("TODO - IMPLEMENT")
}

def giveMeSomeJson(x: JSONObject): Unit = ...

giveMeSomeJson(json"{ name: $name, id: $id }")
```

In this example, we're attempting to create a JSON literal syntax using string interpolation. The JsonHelper implicit class must be in scope to use this syntax, and the json method would need a complete implementation. However, the result of such a formatted string literal would not be a string, but a JSONObject.

When the compiler encounters the literal json"{ name: $name, id: $id }" it rewrites it to the following expression:

```
new StringContext("{ name: ", ", id: ", " }").json(name, id)
```

The implicit class is then used to rewrite it to the following:

```
new JsonHelper(new StringContext("{ name: ", ", id: ", " }")).json(name, id)
```

So, the json method has access to the raw pieces of strings and each expression as a value. A simple (buggy) implementation of this method could be:

```
implicit class JsonHelper(val sc: StringContext) extends AnyVal {
  def json(args: Any*): JSONObject = {
    val strings = sc.parts.iterator
    val expressions = args.iterator
    var buf = new StringBuffer(strings.next)
    while(strings.hasNext) {
      buf append expressions.next
      buf append strings.next
    }
    parseJson(buf)
  }
}
```

Each of the string portions of the processed string are exposed in the StringContext's parts member. Each of the expression values is passed into the json method's args parameter. The json method takes this and generates a big string which it then parses into JSON. A more sophisticated implementation could avoid having to generate this string and simply construct the JSON directly from the raw strings and expression values.

34. Write a Producer and Combiner code in scala
35. What is monad

The simplest way to define a monad is to relate it to a wrapper. Any class object is taken wrapped with a monad in Scala. Just like you wrap any gift or present into a shiny wrapper with ribbons to make them look attractive, Monads in Scala are used to wrap objects and provide two important operations -

- Identity through "unit" in Scala
- Bind through "flatMap" in Scal''"

'Hive'
----

1. What is Difference between partition and bucketing

Partitioning and Bucketing of tables is done to improve the query performance. Partitioning

```
    helps execute queries faster, only if the partitioning scheme has some common range
    filtering i.e. either by timestamp ranges, by location, etc. Bucketing does not work by
    default.
    Partitioning helps eliminate data when used in WHERE clause. Bucketing helps organize data
    inside the partition into multiple files so that same set of data will always be written in
    the same bucket. Bucketing helps in joining various columns.
    In partitioning technique, a partition is created for every unique value of the column and
    there could be a situation where several tiny partitions may have to be created. However,
    with bucketing, one can limit it to a specific number and the data can then be decomposed in
     those buckets.


2. what is different join operations avaialble in Hive

    JOIN-  It is very similar to Outer Join in SQL
    FULL OUTER JOIN - This join Combines the records of both the left and right outer tables.
    Basically, that fulfill the join condition.
    LEFT OUTER JOIN- Through this Join, All the rows from the left table are returned even if
    there are no matches in the right table.
    RIGHT OUTER JOIN - Here also, all the rows from the right table are returned even if there
    are no matches in the left table.


3. What is static and Dynamic partition

    Partitioning in Hive helps prune the data when executing the queries to speed up processing.
     Partitions are created when data is inserted into the table. In static partitions, the name
     of the partition is hardcoded into the insert statement whereas in a dynamic partition,
    Hive automatically identifies the partition based on the value of the partition field.
Based on how data is loaded into the table, requirements for data and the format in which data
is produced at source- static or dynamic partition can be chosen. In dynamic partitions the
complete data in the file is read and is partitioned through a MapReduce job based into the
tables based on a particular field in the file. Dynamic partitions are usually helpful during
ETL flows in the data pipeline.
When loading data from huge files, static partitions are preferred over dynamic partitions as
they save time in loading data. The partition is added to the table and then the file is moved
into the static partition. The partition column value can be obtained from the file name without
 having to read the complete file.

SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode = nonstrict;


4. What is Different Join
    a. Map Side join

        In Apache Hive, there is a feature that we use to speed up Hive queries. Basically, that
         feature is what we call Map join in Hive. Map Join in Hive is also Called Map Side Join
         in Hive. However, there are many more insights of Apache Hive Map join. So, in this
        Hive Tutorial, we will learn the whole concept of Map join in Hive. It includes
        Parameters, limitations of Map Side Join in Hive, Map Side Join in Hive Syntax. Moreover
        , we will see several Map Join in hive examples to understand well.

        here is one more join available that is Common Join or Sort Merge Join. However, there
        is a major issue with that it there is too much activity spending on shuffling data
        around. So, as a result, that slows the Hive Queries. Hence, to speed up the Hive
        queries, we can use Map Join in Hive. Also, we use Hive Map Side Join since one of the
        tables in the join is a small table and can be loaded into memory. So that a join could
        be performed within a mapper without using a Map/Reduce step.

        Parameters of Hive Map Side Join
        a. hive.auto.convert.join
        b. Hive.auto.convert.join.noconditionaltask

        Limitations of Map Join in Hive
```

```
        Below are some limitations of Map Side join in Hive:

At First, the major restriction is, we can never convert Full outer joins to map-side joins.
However, it is possible to convert a left-outer join to a map side join in hive. However, only
possible since the right table that is to the right side of the join conditions, is lesser than
25 MB in size.
Also, we can convert a right-outer join to a map side join in hive. Similarly, only possible if
the left table size is lesser than 25 MB.
5. How to Identify Hive Map Join
Basically, we will see Hive Map Side Join Operator just below Map Operator Tree while using
EXPLAIN command.

Other
Although, we can use the hint to specify the query using Map Join in Hive. Hence, below an
example shows that smaller table is the one put in the hint, and force to cache table B manually.

Select /*+ MAPJOIN(b) */ a.key, a.value from a join b on a.key = b.key

For Example,

hive> set hive.auto.convert.join=true;
hive> set hive.auto.convert.join.noconditionaltask=true;
hive> set hive.auto.convert.join.noconditionaltask.size=20971520
hive> set hive.auto.convert.join.use.nonstaged=true;
hive> set hive.mapjoin.smalltable.filesize = 30000000;

    b. Bucket Map Join

        Basically, while the tables are large and all the tables used in the join are bucketed
        on the join columns we use a Bucket Map Join in Hive. In this article, we will cover the
         whole concept of Apache Hive Bucket Map Join. It also includes use cases, disadvantages
        , and Bucket Map Join example which will enhance our knowledge.

        In Apache Hive, while the tables are large and all the tables used in the join are
        bucketed on the join columns we use Hive Bucket Map Join feature. Moreover, one table
        should have buckets in multiples of the number of buckets in another table in this type
        of join.

        Basically, Join is done in Mapper only.  However, let’s understand it in this way, the
        mapper processing bucket 1 for table A will only fetch bucket 1 of table B.

        Use Case
        To be more specific we use this feature with several scenarios. Like:

        i. While all tables are large.
        ii. Also, while all tables are bucketed using the join columns.
        iii. Moreover, while The number of buckets in one table is a multiple of the number of
        buckets in the other table.
        iii. Also, when all tables are not sorted.

    Disadvantages of Bucket Map Join in Hive
        The major disadvantage of using Bucket Map Join is, here tables need to be bucketed in
        the same way how the SQL joins. That implies we can not use it for other types of SQLs.

    c. Skew Join

        Basically, when there is a table with skew data in the joining column, we use skew join
        feature. On defining what is skewed table, it is a table that is having values that are
        present in large numbers in the table compared to other data. However, while the rest of
         the data is stored in a separate file Skew data is stored in a separate file.

        Parameter
        However, to be set for a Hive skew join we need the following parameter:

        set  hive.optimize.skewjoin=true;
        set hive.skewjoin.key=100000;
```

```
            How Hive Skew Join Works
        However, let’s assume if table A join B, and A has skew data “1” in joining column.
        At First store, the rows with key 1 in an in-memory hash table and read B. Further to
        read A  run a set of mappers. Afterward, do the following:

        Make sure use the hashed version of B to compute the result since it has key 1.
        Then, send all other keys to a reducer which does the join. Basically, from a mapper,
        this reducer will get rows of B also.
        Learn Hadoop from Industry Experts
        Hence, as a result, we end up reading only B twice. Basically, that implies that the
        skewed keys in A are only read and processed by the Mapper. Also, they are not sent to
        the reducer. Moreover, remaining keys in A go through only a single Map/Reduce.
        However, the assumption is that B has few rows with keys which are skewed in A. Hence,
        in this way these rows can be loaded into the memory.

        Skew Join - Use Case
        Basically, on the joining column, one table has huge skew values.
        Let Explore Joins in Hive with Examples

        Disadvantages of Skew Join in Hive
        Here, are some Limitations of Hive Skew Join are discussed:

        So, the major disadvantage of it is One table is read twice here.
        Moreover, it is necessary that users should be aware of the skew key.

    d. Sort Merge Bucket Join

        In Hive, while each mapper reads a bucket from the first table and the corresponding
        bucket from the second table, in SMB join. Basically, then we perform a merge sort join
        feature. Moreover, we mainly use it when there is no limit on file or partition or table
         join. Also, when the tables are large we can use Hive Sort Merge Bucket join. However,
        using the join columns, all join the columns are bucketed and sorted in SMB. Although,
        make sure in SMB join all tables should have the same number of buckets.

        How Hive SMB Works
        Basically, in Mapper, only Join is done. Moreover, all the buckets are joined with each
        other at the mapper which are corresponding.

        Use Case of Sort Merge Bucket Join in Hive
        There are several scenarios when we can use Hive Sort Merge Bucket Join:

        While all tables are Large.
        Also, while all tables are bucketed using the join columns.
        While by using the join columns, Sorted.
        Also, when the number of buckets is same as the number of all tables.
        Read about Map Join in Hive | Map Side Join

        Disadvantages of Sort Merge Bucket Join in Hive
        Following are the limitations of Hive Sort Merge Bucket Join:

        However, in the same way how the SQL joins Tables need to be bucketed. Hence, for other
        types of SQL, it cannot be used.
        Also, it is possible that Partition tables might slow down here.


5. Difference between order by , sort by , distribute by , cluster by


 SORT BY - Data is ordered at each of ‘N’ reducers where the reducers can have overlapping range
  of data.
ORDER BY- This is similar to the ORDER BY in SQL where total ordering of data takes place by
passing it to a single reducer.
DISTRUBUTE BY - It is used to distribute the rows among the reducers. Rows that have the same
distribute by columns will go to the same reducer.
CLUSTER BY- It is a combination of DISTRIBUTE BY and SORT BY where each of the N reducers gets
```

```
non overlapping range of data which is then sorted by those ranges at the respective reducers.



6. How do we intergrate Hive with Spark


7. Difference between Managed Tables and External Tables

    Ans. Managed table,
    The metadata information along with the table data is deleted from the Hive warehouse
    directory if one drops a managed table.
    External table,
    Hive just deletes the metadata information regarding the table. Further, it leaves the table
     data present in HDFS untouched.


8. Different indexes in Hive

    An Index acts as a reference to the records. Instead of searching all the records, we can
    refer to the index to search for a particular record. Indexes maintain the reference of the
    records. So that it is easy to search for a record with minimum overhead. Indexes also speed
     up the searching of data.

        Types of Indexes in Hive
    Compact Indexing
    Bitmap Indexing
    Bit map indexing was introduced in Hive  0.8 and is commonly used for columns with distinct
    values.

    Differences between Compact and Bitmap Indexing
    The main difference is the storing of the mapped values of the rows in the different blocks.
     When the data inside a Hive table is stored by default in the HDFS, they are distributed
    across the nodes in a cluster. There needs to be a proper identification of the data, like
    the data in block indexing. This data will be able to identity which row is present in which
     block, so that when a query is triggered it can go directly into that block. So, while
    performing a query, it will first check the index and then go directly into that block.

    Compact indexing stores the pair of indexed column’s value and its blockid.

    Bitmap indexing stores the combination of indexed column value and list of rows as a bitmap.

9. How to create a Schema for the Data in Hive

    create table hive_table
    (
    )
    fields terminatted by ','
    lines terminated by '\n'
    ;


10. What are different Data types in Hive


    1. Numeric Types
        TINYINT (1-byte signed integer, from -128 to 127)
        SMALLINT (2-byte signed integer, from -32,768 to 32,767)
        INT (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)
        BIGINT (8-byte signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,
        807)
        FLOAT (4-byte single precision floating point number)
        DOUBLE (8-byte double precision floating point number)
        DECIMAL (Hive 0.13.0 introduced user definable precision and scale)
    2. Date/Time Types
        TIMESTAMP
```

```
        DATE
    3. String Types
        STRING
        VARCHAR
        CHAR
    4. Misc Types
        BOOLEAN
        BINARY

    5. Complex Types
        arrays: ARRAY<data_type>
        maps: MAP<primitive_type, data_type>
        structs: STRUCT<col_name : data_type [COMMENT col_comment], ...>
        union: UNIONTYPE<data_type, data_type, ...>


11. How to Select Complex Data Types in Hive

    --arrays
    select ename,subordinates[0] from employees;

    --maps
    select ename,deductions["Federal Taxes"] from employees;

    --structs
    select ename,address.state,address.city,address.street,address.zip from employees;

12. How to create Partition Table for Date column





13. Why Hive is not suitable for OLTP Applications

    Hive is not suitable for OLTP systems because it does not provide insert and update function
     at the row level.

14. What is Metastore in Hive & What is the Metastore in that you used. And How do you configure

    Basically, to store the metadata information in Hive we use Metastore. Though, it is
    possible by using RDBMS and an open source ORM (Object Relational Model) layer called Data
    Nucleus. That converts the object representation into the relational schema and vice versa.

    Local Metastore:

    It is the metastore service runs in the same JVM in which the Hive service is running and
    connects to a database running in a separate JVM. Either on the same machine or on a remote
    machine.

    Remote Metastore:

    In this configuration, the metastore service runs on its own separate JVM and not in the
    Hive service JVM.

    Basically, hive-site.xml file has to be configured with the below property, to configure
    metastore in Hive –
    hive.metastore.uris
    thrift: //node1 (or IP Address):9083
    IP address and port of the metastore host

15. When you should use Sort by instead of Order by

    Despite ORDER BY we should use SORT BY. Especially while we have to sort huge datasets. The
    reason is SORT BY clause sorts the data using multiple reducers. However, ORDER BY sorts all
```

```
 of the data together using a single reducer. Hence, using ORDER BY will take a lot of time
to execute a large number of inputs.

16. What is Partitioning and when do you perform Partitioning

    Basically, for the purpose of grouping similar type of data together on the basis of column
    or partition key, Hive organizes tables into partitions. Moreover, to identify a particular
    partition Each Table can have one or more partition keys. On defining Partition, in other
    words, it is a sub-directory in the table directory.

    However, in a Hive table, Partitioning provides granularity. Hence, by scanning only
    relevant partitioned data instead of the whole dataset it reduces the query latency.

    Dynamic partitioning values for partition columns are known in the runtime. In other words,
    it is known during loading of the data into a Hive table.
    Usage:
    While we Load data from an existing non-partitioned table, in order to improve the sampling.
     Thus it decreases the query latency.
    Also, while we do not know all the values of the partitions beforehand. Thus, finding these
    partition values manually from a huge dataset is a tedious task.



17. What is bucketing and when do you use bucketing

    Basically, for performing bucketing to a partition there are two main reasons:
    A map side join requires the data belonging to a unique join key to be present in the same
    partition.
    It allows us to decrease the query time. Also, makes the sampling process more efficient.

    However, by using the formula: hash_function (bucketing_column) modulo (num_of_buckets) Hive
     determines the bucket number for a row. Basically, hash_function depends on the column data
     type. Although, hash_function for integer data type will be:
    hash_function (int_type_column)= value of int_type_column


18. Explain Hive Indexing

    An Index acts as a reference to the records. Instead of searching all the records, we can
    refer to the index to search for a particular record. Indexes maintain the reference of the
    records. So that it is easy to search for a record with minimum overhead. Indexes also speed
     up the searching of data.

    Hive is a data warehousing tool present on the top of Hadoop, which provides the SQL kind of
     interface to perform queries on large data sets. Since Hive deals with Big Data, the size
    of files is naturally large and can span up to Terabytes and Petabytes. Now if we want to
    perform any operation or a query on this huge amount of data it will take large amount of
    time.

    In a Hive table, there are many numbers of rows and columns. If we want to perform queries
    only on some columns without indexing, it will take large amount of time because queries
    will be executed on all the columns present in the table.

    The major advantage of using indexing is; whenever we perform a query on a table that has an
     index, there is no need for the query to scan all the rows in the table. Further, it checks
     the index first and then goes to the particular column and performs the operation.

    So if we maintain indexes, it will be easier for Hive query to look into the indexes first
    and then perform the needed operations within less amount of time.

    When to use Indexing?
    Indexing can be use under the following circumstances:

    If the dataset is very large.
    If the query execution is more amount of time than you expected.
    If a speedy query execution is required.
```

```
    When building a data model.
    Indexes are maintained in a separate table in Hive so that it won't affect the data inside
    the table, which contains the data. Another major advantage for indexing in Hive is that
    indexes can also be partitioned depending on the size of the data we have.

    Types of Indexes in Hive
    Compact Indexing
    Bitmap Indexing
    Bit map indexing was introduced in Hive  0.8 and is commonly used for columns with distinct
    values.

        Differences between Compact and Bitmap Indexing
        The main difference is the storing of the mapped values of the rows in the different
        blocks. When the data inside a Hive table is stored by default in the HDFS, they are
        distributed across the nodes in a cluster. There needs to be a proper identification of
        the data, like the data in block indexing. This data will be able to identity which row
        is present in which block, so that when a query is triggered it can go directly into
        that block. So, while performing a query, it will first check the index and then go
        directly into that block.

        Compact indexing stores the pair of indexed column's value and its blockid.

        Bitmap indexing stores the combination of indexed column value and list of rows as a
        bitmap.

        Let's now understand what is bitmap?

        A bitmap is is a type of memory organization or image file format used to store digital
        images so with this meaning of bitmap, we can redefine bitmap indexing as given below.

        "Bitmap index stores the combination of value and list of rows as a digital image."

        The following are the different operations that can be performed on Hive indexes:

        Creating index
        Showing index
        Alter index
        Dropping index
        Creating Index in Hive
        Syntax for creating a compact index in Hive is as follows:

        CREATE INDEX index_name
        ON TABLE table_name (columns,....)
        AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
        WITH DEFERRED REBUILD;
        Here, in the place of index_name we can give any name of our choice, which will be the
        table's INDEX NAME.
        In the ON TABLE line, we can give the table_name for which we are creating the index and
         the names of the columns in brackets for which the indexes are to be created. We should
         specify the columns which are available only in the table.
        The org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' line specifies that a
        built in CompactIndexHandler will act on the created index, which means we are creating
        a compact index for the table.
        The WITH DEFERRED REBUILD statement should be present in the created index because we
        need to alter the index in later stages using this statement.
        This syntax will create an index for our table, but to complete the creation, we need to
         complete the REBUILD statement. For this to happen, we need to add one more alter
        statement. A MapReduce job will be launched and the index creation is now completed.

        Hadoop

        ALTER INDEX index_nam on table_name REBUILD;

        This ALTER statement will complete our REBUILDED index creation for the table.

        Examples – Creating Index
```

```
In this section we will first execute the hive query on non-indexed table and will note
down the time taken by query to fetch the result.

In the second part, we will be performing the same query on indexed table and then will
compare the time taken by query to fetch the result with the earlier case.

We will be demonstrating this difference of time with practical examples.

In first scenario we are performing operations on non-indexed table.

Let’s create a normal managed table to contain the olympic dataset first.

Table Creation
create table olympic(athelete STRING,age INT,country STRING,year STRING,closing STRING,
sport STRING,gold INT,silver INT,bronze INT,total INT) row format delimited fields
terminated by '\t' stored as textfile;
Here we are creating a table with name ‘olympic’. The schema of the table is as
specified and the data inside the input file is delimited by tab space.

At the end of the line, we have specified ‘stored as textfile’, which means we are using
 a TEXTFILE format.

You can check the schema of your created table using the command ‘describe olympic;’

We can load data into the created table as follows:

load data local inpath ‘path of your file‘into table olympic;



We have successfully loaded the input file data into the table which is in the TEXTFILE
format.

Let’s perform an Average operation on this ‘olympic’ data. Let’s calculate the average
age of the athletes using the following command:

SELECT AVG(age) from olympic;



Here we can see the average age of the athletes to be 26.405433646812956 and the time
for performing this operation is 21.08 seconds.

Now, let’s create the index for this table:

CREATE INDEX olympic_index
ON TABLE olympic (age)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD;



ALTER INDEX olympic_index on olympic REBUILD;



Here we have created an index for the ‘olympic’ table on the age column. We can view the
 indexes created for the table by using the below command:

show formatted index on olympic;



We can see the indexes available for the ‘olympic’ table in the above image.

Now, let’s perform the same Average operation on the same table.
```

```
    We have now got the average age as 26.405433646812956, which is same as the above, but
    now the time taken for performing this operation is 17.26 seconds, which is less than
    the above case.

    Now we know that by using indexes we can reduce the time of performing the queries.

    Can we have different indexes for the same table?
    Yes! We can have any number of indexes for a particular table and any type of indexes as
     well.

    Let's create a Bitmap index for the same table:

    CREATE INDEX olympic_index_bitmap
    ON TABLE olympic (age)
    AS 'BITMAP'
    WITH DEFERRED REBUILD;

    ALTER INDEX olympic_index_bitmap on olympic REBUILD;

19. Explain Different types of Joins in Hive -- Duplicate
20. Explain --Duplicate
    a. Bucket Map Join
    b. Skew Join
    c. Sort Merge Bucket Join
21. Explain SORT BY, ORDER BY, DISTRIBUTE BY and CLUSTER BY with Example
22. How do process query for
    a. XML

    1) create table xmlsample_guru(str string);
    2) load data local inpath '/home/hduser/test.xml' overwrite  into table xmlsample_guru;
    3) select xpath(str,'emp/ename/text()'), xpath(str,'emp/esal/text()') from
    xmlsample_guru;

    b. Json
    1) create table json_guru(str string);
    2) load data inpath 'home/hduser/test.json' into table json_guru;
    3) select * from json1;
    4) select get_json_object(str,'$.ecode') as ecode, get_json_object(str,'$.ename') as
    ename ,get_json_object(str,'$.sal') as salary from json_guru;

    c. CSV

    CREATE TABLE AllstarFull (playerID string,yearID string,gameNum string,gameID string,
    teamID string,lgID string,GP string,startingPos string) row format delimited fields
    terminated by ',' stored as textfile;

    LOAD DATA INPATH '/user/bigdataproject/AllstarFull.csv' OVERWRITE INTO TABLE AllstarFull;

    SELECT * FROM AllstarFull;


23. What are complex data types and how do you query Hive Collections

    hive> create table employees ( ename string,salary float,subordinates array<string>,
    deductions map<string,float>,address struct<street:string,city:string,state:string,zip:int>)
    > row format delimited
    > fields terminated by '\001'
    > collection items terminated by '\002'
    > map keys terminated by '\003';


    load data local inpath '/home/cloudera/Downloads/employees.txt'
    OVERWRITE into table employees;
```

```
    --arrays
    select ename,subordinates[0] from employees;

    --maps
    select ename,deductions["Federal Taxes"] from employees;

    --structs
    select ename,address.state,address.city,address.street,address.zip from employees;

24. Explain What are the Optimization Technique Avaialble in Hive

    Types of Query Optimization Techniques in Hive
        a. Tez-Execution Engine in Hive
        b. Usage of Suitable File Format in Hive
        c. Hive Partitioning
        d. Bucketing in Hive
        e. Vectorization In Hive
        f. Cost-Based Optimization in Hive (CBO)
        g. Hive Indexing

25. Explain Views in Hive

    A view allows a query to be saved and treated like a table. It is a logical construct, as it
     does not store data like a table. In other words, materialized views are not currently
    supported by Hive.

    When a query references a view, the information in its definition is combined with the rest
    of the query by Hive’s query planner. Logically, you can imagine that Hive executes the view
     and then uses the results in the rest of the query.

    Views to Reduce Query Complexity
    When a query becomes long or complicated, a view may be used to hide the complexity by
    dividing the query into smaller, more manageable pieces; similar to writing a function in a
    programming language or the concept of layered design in software. Encapsulating the
    complexity makes it easier for end users to construct complex queries from reusable parts.
    For example, consider the following query with a nested subquery:

    It is common for Hive queries to have many levels of nesting. In the following example, the
    nested portion of the query is turned into a view:

    CREATE VIEW shorter_join AS
    SELECT * FROM people JOIN cart
    ON (cart.people_id=people.id) WHERE firstname='john';
    Now the view is used like any other table. In this query we added a WHERE clause to the
    SELECT statement. This exactly emulates the original query:

    SELECT lastname FROM shorter_join WHERE id=3;

26. Did you used UDFs in Hive
        -- Yes (But its not written by Me) we used it for Percentile calculations

27. What is Beelime

    HiveServer2 supports a command shell Beeline that works with HiveServer2. It's a JDBC
    client that is based on the SQLLine CLI (http://sqlline.sourceforge.net/). There’s detailed
    documentation of SQLLine which is applicable to Beeline as well.
Replacing the Implementation of Hive CLI Using Beeline
The Beeline shell works in both embedded mode as well as remote mode. In the embedded mode, it
runs an embedded Hive (similar to Hive CLI) whereas remote mode is for connecting to a separate
HiveServer2 process over Thrift. Starting in Hive 0.14, when Beeline is used with HiveServer2,
it also prints the log messages from HiveServer2 for queries it executes to STDERR. Remote
HiveServer2 mode is recommended for production use, as it is more secure and doesn't require
direct HDFS/metastore access to be granted for users.
```

```
28. What version of Hive you used in your organization



29. What is Impala --Not used

30. Explain Different SET Operations in Hive

    set hive.cli.current.print.current.db=true
    set hive.auto.convert.join=true
    set hive.exec.dynamic.partition=true
    set hive.exec.dynamic.partition.mode=nonstrict;
    set mapred.reduce.tasks=50
    set hive.exec.reducers.max=50

31. Why do you drop a External Table

    -- Needs answer


32. Explain Serde in Hive
33. What are File Formats supported by Hive -- Check 44
34. Explain variables in Hive

    hive> set CURRENT_DATE='2012-09-16';
    hive> select * from foo where day >= '${hiveconf:CURRENT_DATE}'
    % hive -hiveconf CURRENT_DATE='2012-09-16' -f test.hql

35. Explain How do you insert Date in Hive Table

    insert into table_name values ();

    insert into new_tables
    select * from table_name;

36. Explain Analytical functions in Hive

    1. count
    2. sum
    3. lead
    4. lag
    5. FIRST_VALUE
    6. ROW_NUMBER
    7. Rank
    8. Dense Rank

37. How do you delete Duplicates in Hive

    insert overwrite table dynpart select distinct * from dynpart;

    1) Create a new table from old table (with same structure).
    2) Copy distinct rows in new table from existing table.
       select col1,col2,col3,col4,max(<duplicate column>) as <name of duplicate column> from <
       table name> group by col1,col2,col3,col4;
    3) Delete old table.
    4) Rename new table to old one.

38. Explain Architecture of Hive

    There are several components of Hive Architecture. Such as -
User Interface - Basically, it calls the execute interface to the driver. Further, driver
creates a session handle to the query. Then sends the query to the compiler to generate an
execution plan for it.
Metastore - It is used to Send the metadata to the compiler. Basically, for the execution of the
 query on receiving the send MetaData request.
Compiler- However, it generates the execution plan. Especially, that is a DAG of stages where
```

each stage is either a metadata operation, a map or reduce job or an operation on HDFS.

39. What is Apache HCatalog

Hcatalog can be used to share data structures with external systems. Hcatalog provides access to hive metastore to users of other tools on Hadoop so that they can read and write data to hive's data warehouse.

40. What is Hive Current Version and What is Hive stable Version

41. Difference between SQL and HQL

SQL : It supports DML
HQL : It doesn not support DML

42. How do you pull the Oracle data into Hive

```
sqoop import \
--connect jdbc:mysql://localhost/dualcore \
--username training \
--password training \
--m 1 \
--target-dir /queryresult \
--table employees \
--hive-import
```

43. How to integrate Hive with Spark

```
  val sparkSession = SparkSession.builder
  .master("local")
  .appName("demo")
.enableHiveSupport()
  .getOrCreate()

  sparkSession.sqlContext.sql("INSERT INTO TABLE students VALUES ('Rahul','Kumar'), ('abc','xyz'
  )")
```

44. Hive File types

File formats in Hive
a. Row Based File format
b. column based file format

a. Row based file format
1. Text file Format
2. Sequence File Format
3. Avro File Format

b. Column Based File format
1. RC File
2. ORC File
3. Paruet

a.1. Text file Format :
It is the default format of Hive, It is a human readable file format.
Text file format doesnot allow compression technique. It has interoperable to HDFS and non HDFS. It takes huge space.
Example : tab separated file, comman separated , space separated

How to create text file format :

```
            create table emp (
                empno int,ename string,salary float)
            stored as TextFile;

        describe formatted table_name : org.apache.hadoop.hive.ql.io.
        HiveIgnoreKeyTextOutputFormat.


    a.2 Sequence File Format :
        It is the binary file format and it is a row based file format. It is not a
        human readable format.
        Sequence file format binary or images. Sequence file format supports compression
        . performance wise very good.

        Drawback : Poor interoperable. It supports only HDFs .


    create table emp_sequence
    (eno int,
    ename string,
    salary int,
    gendar string,
    dno int
    ) stored as SEQUENCEFILE;

      you cannot extract the schema

    a.3. AVRO

      AVRO file format is sequence file format and on top of the data machine creates a
      schema.

      AVRO tool creates a schema on top of the data.




    Step 1:
    sqoop import-all-tables \
    --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
    --username retail_dba \
    --password cloudera \
    --warehouse-dir /user/hive/warehouse/retail_stage.db \
    --compress \
    --compression-codec snappy \
    --as-avrodatafile
    -m 1;


    Step 2:
    hadoop fs -get /user/hive/warehouse/retail_stage.db/orders/part-m-00000.avro
    avro-tools getschema part-m-00000.avro > orders.avsc
    hadoop fs -mkdir /user/hive/schemas
    hadoop fs -ls /user/hive/schemas/order
    hadoop fs -copyFromLocal orders.avsc /user/hive/schemas/order

    Launch HIVE using 'hive' command in a separate terminal

    Below HIVE command will create a table pointing to the avro data file for orders data

    create external table orders_sqoop
    STORED AS AVRO
    LOCATION '/user/hive/warehouse/retail_stage.db/orders'


    Column Based format .
```

```
        1. RC File format (Row column) --
                The Default column based file format is RC File format .
                If we are using RC file format. All the column formats by defaultly string.
                It occupies more space.
                RC File format supports compression

                Drawback is : Poor performance and less interoperability


        Example :
        create table emp_rc (
        no int,
        ename string,
        salary int,
        gendar string,
        dno int
        ) stored as RCFILE;


        2. ORC File Format :
                Horton works introduced improved version of RC File.
                Based on Input data is stored as that data type instead of all string.
                Drawback is it only support Hadoop.

        3. Parquet

                Check more about parquet

                1. If use case is more about reading the data -- AVRO
                2. If use case is more about writing the data of single column -- PARQUET
                3. If use case is more about writing data to many columns -- AVRO

                Mostly we use AVRO in realtime

                Spark default file format is parquet.



'Sqoop'
--------
1. How to Import Query data into HDFS

    sqoop import \
    --connect jdbc:oracle:thin:@//localhost:1521/xe \
    --username scott \
    --password tiger \
    --table EMP \
    --warehouse-dir /user/cloudera/sqoop_dir     or --target-dir /etl/input/cities

2. How to Import Data from Oracle to Hive Table or Hive Partitions


sqoop import \
--connect jdbc:oracle:thin:@//HYRDSLVM0028.es.ad.adp.com:1521/cri02hyd \
--username scott \
--password tiger \
--table EMP \
--hive-import \
--create-hive-table \
--hive-table naravishdb.EMP \
--null-string '\\N' \
--null-non-string '\\N' \


3. How to do incremental import using sqoop
```

```
    sqoop import \
    --connect jdbc:oracle:thin:@//localhost:1521/xe \
    --username scott \
    --password tiger \
    --table SALGRADE \
    --incremental append \
    --check-column GRADE \
    --last-value 5 \
    --warehouse-dir /user/cloudera/sqoop_dir/

4. How to craeate job or store the last value and retrieve in sqoop

    sqoop job \
    --create salgrade \
    -- \
    import \
    --connect jdbc:oracle:thin:@//localhost:1521/xe \
    --username scott \
    --password tiger \
    --table SALGRADE \
    --incremental append \
    --check-column GRADE \
    --last-value 6

5. How to set the boundry in sqoop

    --boundary clause

6. How to import data into HBase

7. Boundary Query
        -- Boundary Condition
      sqoop import \
    --connect jdbc:oracle:thin:@//localhost:1521/xe \
    --username scott \
--password tiger \
--query 'SELECT normcities.id, \
countries.country, \
normcities.city \
FROM normcities \
JOIN countries USING(country_id) \
WHERE $CONDITIONS' \
--split-by id \
--target-dir cities \
--boundary-query "select min(id), max(id) from normcities"'


8. $CONDITIONS

    Sqoop performs highly efficient data transfers by inheriting Hadoop’s parallelism.

            To help Sqoop split your query into multiple chunks that can be transferred in
            parallel, you need to include the $CONDITIONS placeholder in the where clause of
            your query.
            To help Sqoop split your query into multiple chunks that can be transferred in
            parallel, you need to include the $CONDITIONS placeholder in the where clause of
            your query.
            Sqoop will automatically substitute this placeholder with the generated conditions
            specifying which slice of data should be transferred by each individual task.
            Sqoop will automatically substitute this placeholder with the generated conditions
            specifying which slice of data should be transferred by each individual task.
            While you could skip $CONDITIONS by forcing Sqoop to run only one job using the
            --num-mappers 1 param eter, such a limitation would have a severe performance
            impact.
            While you could skip $CONDITIONS by forcing Sqoop to run only one job using the
```

```
            --num-mappers 1 param eter, such a limitation would have a severe performance
            impact.

9. --where

    To specify conditions while import or export

        10. Append and overwrite Directo
ry (overwrite doesnot exist, we need to handle separatlely in shell)

11. How to do Incremental load or delta load
    --Using Last value

12. Insert/update in Sqoop Incremental
    Why update not work in sqoop

    to check

13. Integeration of Hive with Sqoop

14. How you query using sqoop

    sqoop eval --connect --query "select count(0) from emp"

15. How to pull all the tables using sqoop

    --import-all-tables

16. What are file formats supported by sqoop

    sqoop su

    Newer Version of Sqoop support file formats like
    1. sequence file format
    2. Avro File format
    3. Parquet file format

17. Does Sqoop supports CLOB Columns

    sqoop import \
    -Dmapred.job.queue.name=default \
    -connect jdbc:oracle:thin:@hostname:port/port \
    -username Xxxxxx \
    -password XXXXXX \
    -query “SELECT * FROM tablename WHERE \$CONDITIONS” \
    -hive-drop-import-delims \
    -map-column-java column1=String,column2=String \
    -m 8 \
    -hive-import \
    -hive-table tablename \
    -target-dir /user/hdfs/ \
    -fields-terminated-by ’01’ \
    -split-by id;

18. Different Options avaialbe in sqoop --> Same as 38
19. What is better sqoop or Spark pull

        Spark is better than sqoop for Data Extraction as spark works on In-memory.
        Sqoop works with I/o

20. How you do incremental pull using sqoop job

21. How to Handle Null in sqoop import

    --null-string           --> Null String
    --null-non-string       --> Null for non strings
```

```
22. Explain --append option in sqoop

23. Explain free form query in sqoop

    Use --query

24. Difference between --target-dir --warehouse-dir

    --target-dir Mainly used Importing a Single Table into HDFS
    --always target directory looking for a new directory in HDFS

    --warehouse-dir
    If you want to import all the tables of schema we use



25. How to store and use last value in sqoop job

    .If an incremental import is run from the command line,the value which should be specified
    as --last-value in a subsequent incremental import will be printed to the screen for your
    reference.

    If an incremental import is run from a saved job, this value will be retained in the saved
    job. Subsequent runs of sqoop job  will continue to import only newer rows than those
    previously imported.


26. How to used password file

    sqoop import --connect jdbc:mysql://localhost:3306/db --username bhavesh --password-file
    /pwd --table t1 --target-dir '/erp/test'

27. where you should copy the jars

    cp /usr/lib/hive/lib/mysql*.jar /usr/lib/hadoop/lib

28. How to exclude table in import all

    --exclude-table table_list

29. How to increase number of mappers

    --m 10
    --num-of_mapper

30. how to do compression

    --compress   \

31. Is it possible to update record using sqoop

    With insert mode, records exported by Sqoop are appended to the end of the target table.
    Sqoop also provides an update mode that you can use by providing the -update-key <column(s)>
     command line argument. This action causes Sqoop to generate a SQL UPDATE statement to run
    on the RDBMS or data warehouse.

Assume that you want to update a three-column table with data stored in the HDFS file /user/my-
hdfs-file. The file contains this data:

100, 1000, 2000
The following abbreviated Sqoop export command generates the corresponding SQL UPDATE statement
on your database system:

$ sqoop export (Generic Arguments)
  --table target-relational-table
```

```
  --update-key column1
  --export-dir /user/my-hdfs-file
  ...
Generates => UPDATE target-relational-table SET
                    column2=1000,column3=2000
              WHERE column1=100;

32. Export and Import Data from and to Oracle

    sqoop import \
    --connect jdbc:oracle:thin:@//HYRDSLVM0028.es.ad.adp.com:1521/cri02hyd \
    --username hr \
    --password hr \
    --table JOBS

    sqoop export \
--connect jdbc:oracle:thin:@//localhost:1521/xe \
--username scott \
--password tiger \
--table EMP_DEPT \
--export-dir '/user/cloudera/emp_dept.txt'


33. Export and Import Data from and to Hive

    sqoop import \
--connect jdbc:mysql://mysql.example.com/sqoop \
--username sqoop \
--password sqoop \
--table cities \
--hive-import \
--hive-partition-key day \
--hive-partition-value "2013-05-22"

sqoop export \
--connect jdbc:oracle:thin:@//HYRDSLVM0028.es.ad.adp.com:1521/cri02hyd \
--username scott \
--password tiger \
--table EMP \
--null-string '\\N' --input-null-string '\\N' \
--export-dir 'naravish.db/emp/part*' \
--fields-terminated-by '\001'

34. Export and Import Data from and to Hbase

    sqoop import \
--connect jdbc:mysql://mysql.example.com/sqoop \
--username sqoop \
--password sqoop \
--table cities \
--hbase-table cities \
--column-family world

sqoop export \
--connect jdbc:oracle:thin:@//HYRDSLVM0028.es.ad.adp.com:1521/cri02hyd \
--username scott \
--password tiger \
--table EMP2 \
--input-null-string '\\N' \
--input-null-non-string '\\N' \
--export-dir 'naravish.db/emp_permanent_hbase/*' \
--fields-terminated-by ','


36. The nine functions of Sqoop?
    A.  Full Load
```

```
    B.  Incremental Load
    C.  Parallel import/export
    D.  Import results of SQL query
    E.  Compression
    F.  Connectors for all major RDBMS Databases
    G.  Kerberos Security Integration
    H.  Load data directly into Hive/Hbase
    I.  Support for Accumulo
37. Default number of parallel jobs

    Defaultly sqoop does 4 parallel jobs

38. Explain

    --append                --> creates a new part file
    --as-avrodatafile       --> Saves data as avro Data file
    --as-sequencefile       --> Saves data as s
    --as-textfile           --> Saves as Textfile
    --boundary-query        --> Query to specify min and max value
    --columns               --> To pull only certain columns
    --direct                --> --direct - Use direct import fast path
    --direct-split-size     -->
    --inline-lob-limit      -->
    ---m                    --> Num of Mappers
    --e,--query             --> Customer Query
    --split-by              --> column value that should be used to min and max value incase of
    no PK
    --table                 --> Table Name
    --target-dir            --> Target Directory
    --warehouse-dir         --> Incase of importing import-all tables
    --where                 --> Where clause to import data
    --compress              --> Compression
    --compression-codec     --> Dont know
    --null-string           --> Null String
    --null-non-string       --> Null for non strings

'HDFS'

1. What is Data Locality

    Data locality refers to the ability to move the computation close to where the actual data
    resides on the node, instead of moving large data to computation. This minimizes network
    congestion and increases the overall throughput of the system.

2. Difference between 1.0 vs 2.0

    Hadoop V.1.x Components

        Apache Hadoop V.1.x has the following two major Components

        HDFS (HDFS V1)
        MapReduce (MR V1)
        In Hadoop V.1.x, these two are also know as Two Pillars of Hadoop.

        hadoop1.x-components

        Hadoop V.2.x Components

        Apache Hadoop V.2.x has the following three major Components

        HDFS V.2
        YARN (MR V2)
        MapReduce (MR V1)
        In Hadoop V.2.x, these two are also know as Three Pillars of Hadoop.

        hadoop2.x-components
```

```
Hadoop 1.x Limitations

Hadoop 1.x has many limitations or drawbacks. Main drawback of Hadoop 1.x is that
MapReduce Component in it’s Architecture. That means it supports only MapReduce-based
Batch/Data Processing Applications.

Hadoop 1.x has the following Limitations/Drawbacks:

It is only suitable for Batch Processing of Huge amount of Data, which is already in
Hadoop System.
It is not suitable for Real-time Data Processing.
It is not suitable for Data Streaming.
It supports upto 4000 Nodes per Cluster.
It has a single component : JobTracker to perform many activities like Resource
Management, Job Scheduling, Job Monitoring, Re-scheduling Jobs etc.
JobTracker is the single point of failure.
It does not support Multi-tenancy Support.
It supports only one Name Node and One Namespace per Cluster.
It does not support Horizontal Scalability.
It runs only Map/Reduce jobs.
It follows Slots concept in HDFS to allocate Resources (Memory, RAM, CPU). It has static
 Map and Reduce Slots. That means once it assigns resources to Map/Reduce jobs, it
cannot re-use them even though some slots are idle.
For Example:- Suppose, 10 Map and 10 Reduce Jobs are running with 10 + 10 Slots to
perform a computation. All Map Jobs are doing their tasks but all Reduce jobs are idle.
We cannot use these Idle jobs for other purpose.

NOTE:- In Summary, Hadoop 1.x System is a Single Purpose System. We can use it only for
MapReduce Based Applications.

Differences between Hadoop 1.x and Hadoop 2.x

If we observe the components of Hadoop 1.x and 2.x, Hadoop 2.x Architecture has one
extra and new component that is : YARN (Yet Another Resource Negotiator).

It is the game changing component for BigData Hadoop System.

New Components and API
As shown in the below diagram, Hadoop 1.x is re-architected and introduced new component
 to solve Hadoop 1.x Limitations.

hadoop1_vs_hadoop2

Hadoop 1.x Job Tracker
As shown in the below diagram, Hadoop 1.x Job Tracker component is divided into two
components:

Resource Manager:-
To manage resources in cluster

Application Master:-
To manage applications like MapReduce, Spark etc.

hadoop1_jobtracker_hadoop2

Hadoop 1.x supports only one namespace for managing HDFS filesystem whereas Hadoop 2.x
supports multiple namespaces.
Hadoop 1.x supports one and only one programming model: MapReduce. Hadoop 2.x supports
multiple programming models with YARN Component like MapReduce, Interative, Streaming,
Graph, Spark, Storm etc.
Hadoop 1.x has lot of limitations in Scalability. Hadoop 2.x has overcome that
limitation with new architecture.
Hadoop 2.x has Multi-tenancy Support, but Hadoop 1.x doesn’t.
Hadoop 1.x HDFS uses fixed-size Slots mechanism for storage purpose whereas Hadoop 2.x
uses variable-sized Containers.
```

Hadoop 1.x supports maximum 4,000 nodes per cluster where Hadoop 2.x supports more than 10,000 nodes per cluster.
How Hadoop 2.x solves Hadoop 1.x Limitations

Hadoop 2.x has resolved most of the Hadoop 1.x limitations by using new architecture.

By decoupling MapReduce component responsibilities into different components.
By Introducing new YARN component for Resource management.
By decoupling component's responsibilities, it supports multiple namespace, Multi-tenancy, Higher Availability and Higher Scalability.
Hadoop 2.x YARN Benefits

Hadoop 2.x YARN has the following benefits.

Highly Scalability
Highly Availability
Supports Multiple Programming Models
Supports Multi-Tenancy
Supports Multiple Namespaces
Improved Cluster Utilization
Supports Horizontal Scalability

3. Explain the Architecture of 2.0

Hadoop 2.x Architecture

Apache Hadoop 2.x or later versions are using the following Hadoop Architecture. It is a Hadoop 2.x High-level Architecture. We will discuss in-detailed Low-level Architecture in coming sections.

hadoop architecture

Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
HDFS stands for Hadoop Distributed File System. It is also know as HDFS V2 as it is part of Hadoop 2.x with some enhanced features. It is used as a Distributed Storage System in Hadoop Architecture.
YARN stands for Yet Another Resource Negotiator. It is new Component in Hadoop 2.x Architecture. It is also know as "MR V2".
MapReduce is a Batch Processing or Distributed Data Processing Module. It is also know as "MR V1" as it is part of Hadoop 1.x with some updated features.
Remaining all Hadoop Ecosystem components work on top of these three major components: HDFS, YARN and MapReduce. We will discuss all Hadoop Ecosystem components in-detail in my coming posts.
When compared to Hadoop 1.x, Hadoop 2.x Architecture is designed completely different. It has added one new component : YARN and also updated HDFS and MapReduce component's Responsibilities.

Hadoop 2.x Major Components

Hadoop 2.x has the following three Major Components:

HDFS
YARN
MapReduce
These three are also known as Three Pillars of Hadoop 2. Here major key component change is YARN. It is really game changing component in BigData Hadoop System.

How Hadoop 2.x Major Components Works

Hadoop 2.x components follow this architecture to interact each other and to work parallel in a reliable, highly available and fault-tolerant manner.

Hadoop 2.x Components High-Level Architecture

```
        hadoop 2 architecture diagram

        All Master Nodes and Slave Nodes contains both MapReduce and HDFS Components.
        One Master Node has two components:
        Resource Manager(YARN or MapReduce v2)
        HDFS
        It's HDFS component is also knows as NameNode. It's NameNode is used to store Meta Data.

        In Hadoop 2.x, some more Nodes acts as Master Nodes as shown in the above diagram. Each
        this 2nd level Master Node has 3 components:
        Node Manager
        Application Master
        Data Node
        Each this 2nd level Master Node again contains one or more Slave Nodes as shown in the
        above diagram.
        These Slave Nodes have two components:
        Node Manager
        HDFS
        It's HDFS component is also knows as Data Node. It's Data Node component is used to
        store actual our application Big Data. These nodes does not contain Application Master
        component.

        Hadoop 2.x Components In-detail Architecture

        hadoop components and architecture

        Hadoop 2.x Architecture Description

        Resource Manager:
        Resource Manager is a Per-Cluster Level Component.
        Resource Manager is again divided into two components:
        Scheduler
        Application Manager
        Resource Manager's Scheduler is :
        Responsible to schedule required resources to Applications (that is Per-Application
        Master).
        It does only scheduling.
        It does care about monitoring or tracking of those Applications.
        Application Master:

        Application Master is a per-application level component. It is responsible for:
        Managing assigned Application Life cycle.
        It interacts with both Resource Manager's Scheduler and Node Manager
        It interacts with Scheduler to acquire required resources.
        It interacts with Node Manager to execute assigned tasks and monitor those task's status.
        Node Manager:

        Node Manager is a Per-Node Level component.
        It is responsible for:
        Managing the life-cycle of the Container.
        Monitoring each Container's Resources utilization.
        Container:

        Each Master Node or Slave Node contains set of Containers. In this diagram, Main Node's
        Name Node is not showing the Containers. However, it also contains a set of Containers.
        Container is a portion of Memory in HDFS (Either Name Node or Data Node).
        In Hadoop 2.x, Container is similar to Data Slots in Hadoop 1.x. We will see the major
        differences between these two Components: Slots Vs Containers in my coming posts.
        NOTE:-

        Resource Manager is Per-Cluster component where as Application Master is per-application
         component.
        Both Hadoop 1.x and Hadoop 2.x Architectures follow Master-Slave Architecture Model.

4. Explain the role of YARN
```

```
    Apache Yarn – "Yet Another Resource Negotiator" is the resource management layer of Hadoop.
    The Yarn was introduced in Hadoop 2.x. Yarn allows different data processing engines like
    graph processing, interactive processing, stream processing as well as batch processing to
    run and process data stored in HDFS (Hadoop Distributed File System). Apart from resource
    management, Yarn also does job Scheduling. Yarn extends the power of Hadoop to other
    evolving technologies, so they can take the advantages of HDFS (most reliable and popular
    storage system on the planet) and economic cluster. To learn installation of Apache Hadoop 2
     with Yarn follows this quick installation guide.

    Apache yarn is also a data operating system for Hadoop 2.x. This architecture of Hadoop 2.x
    provides a general purpose data processing platform which is not just limited to the
    MapReduce. It enables Hadoop to process other purpose-built data processing system other
    than MapReduce. It allows running several different frameworks on the same hardware where
    Hadoop is deployed.

    In this section of Hadoop Yarn tutorial, we will discuss the complete architecture of Yarn.
    Apache Yarn Framework consists of a master daemon known as "Resource Manager", slave daemon
    called node manager (one per slave node) and Application Master (one per application).

    Resource Manager has two Main components

    Scheduler
    Application manager

    Node Manager (NM)
    Application Master (AM)
    Resource Manager Restart
    Yarn Resource Manager High availability

5. What is the Issue with Hadoop 1.0.

    The NameNode is the single point of failure in Hadoop 1.0.
Each cluster has a single NameNode and if that machine is not available, the whole cluster will
be not available.
This impacts the total availability of HDFS in two ways:

For any unplanned event such as machine crashes, the whole cluster is not available until the
Name node is brought up manually.
For planned maintenance such as Hardware or Software upgrades on NameNode would result in
cluster unavailability.
In Hadoop 2.0, HDFS High Availability feature addresses the above problem, by providing an
option to run two NameNodes in the same cluster in an Active/Passive configuration with a hot
standby.
This allows fast Failover to a new NameNode for any machine crashes or administrator initiated
fail-over for any planned maintenance activities.

6. How Name node single point of failure is rectified in Hadoop 2.0

    HDFS High Availability of Namenode is introduced with Hadoop 2. In this two separate
    machines are getting configured as NameNodes, where one NameNode always in working state and
     anther is in standby. Working Name node handling all clients request in the cluster where
    standby is behaving as the slave and maintaining enough state to provide a fast failover on
    Working Name node.

7. Why block size is 128 KB in Hadoop

    HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost of
    seeks. By making a block large enough, the time to transfer the data from the disk can be
    significantly longer than the time to seek to the start of the block. Thus the time to
    transfer a large file made of multiple blocks operates at the disk transfer rate.

    A quick calculation shows that if the seek time is around 10 ms and the transfer rate is 100
     MB/s, to make the seek time 1% of the transfer time, we need to make the block size around
    100 MB. The default is actually 64 MB, although many HDFS installations use 128 MB blocks.
    This figure will continue to be revised upward as transfer speeds grow with new generations
    of disk drives.
```

This argument shouldn't be taken too far, however. Map tasks in MapReduce normally operate on one block at a time, so if you have too few tasks (fewer than nodes in the cluster), your jobs will run slower than they could otherwise.

8. Exaplain
    a. Edit logs
    b. FSImage

9. Explain how fault tolerant is achieved in Hadoop

    Using Replication factor of 3

10. Why Hadoop

    Hadoop Provides 2 Main Important things
    1. HDFS For Distributed storage
    2. Map Reduce for Distributed Processing

    Hadoop Make it possible
    a. Expense is low (Commodity Hardware)
    b. Fast Processing beacouse of Data locality
    c. Fault tolerant (Replication Factor)
    d. Possible Horizontal Scalability

11. Explain Heartbeat in Hadoop

    Heartbeat is the signal that is sent by the datanode to the namenode after the regular interval to time to indicate its presence, i.e. it is alive. NameNode and DataNode do communicate using Heartbeat. If after the certain time of heartbeat, NameNode do not receive any response from DataNode, then that Node is dead.

12. Explain the replication factor in Hadoop

    For example, if the replication factor was set to 3 (default value in HDFS) there would be one original block and two replicas. hdfs-site.xml is used to configure HDFS. ... The block size setting is used by HDFS to divide files into blocks and then distribute those blocks across the cluster.Feb 12, 2018

    If we write a data to block Hadoop takes responsibillity to copy to other 2 data nodes. And it is rack aware.

13. Explain Safe mode in Hadoop

    Safemode in Apache Hadoop is a maintenance state of NameNode. During which NameNode doesn't allow any modifications to the file system. During Safemode, HDFS cluster is in read only and doesn't replicate or delete block.

    In SafeMode NameNode performs collection of block reports from datanodes. NameNode enters safemode automatically during its start up. NameNode leaves Safemode after the DataNodes have reported that most Blocks are available.

    To know the status of Safemode, use command: hadoop dfsadmin –safemode get

    To enter Safemode, use command: bin/hadoop dfsadmin –safemode enter
    To come out of Safemode, use command: hadoop dfsadmin -safemode

14. Explain Small file problem in Hadoop

    Problems with small files and HDFS
A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you're storing small files, then you probably have lots of them (otherwise you wouldn't turn to Hadoop), and the problem is that HDFS can't handle lots of files.

Every file, directory and block in HDFS is represented as an object in the namenode's memory, each of which occupies 150 bytes, as a rule of thumb. So 10 million files, each using a block, would use about 3 gigabytes of memory. Scaling up much beyond this level is a problem with current hardware. Certainly a billion files is not feasible.

Furthermore, HDFS is not geared up to efficiently accessing small files: it is primarily designed for streaming access of large files. Reading through small files normally causes lots of seeks and lots of hopping from datanode to datanode to retrieve each small file, all of which is an inefficient data access pattern.

Problems with small files and MapReduce
Map tasks usually process a block of input at a time (using the default FileInputFormat). If the file is very small and there are a lot of them, then each map task processes very little input, and there are a lot more map tasks, each of which imposes extra bookkeeping overhead. Compare a 1GB file broken into 16 64MB blocks, and 10,000 or so 100KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file.

There are a couple of features to help alleviate the bookkeeping overhead: task JVM reuse for running multiple map tasks in one JVM, thereby avoiding some JVM startup overhead (see the mapred.job.reuse.jvm.num.tasks property), and MultiFileInputSplit which can run more than one split per map.

15. Why Hadoop is less costly

    Because hadoop does not needs high end computing Serves , it relies on Large number of commodity hardwares.

16. Explain Rack Awareness in Hadoop

    Data Replications happens in Different Racks. For Exaple if there there 2 Racks then Hadoop Framework make sure one replication is at least available in each rack

17. Explain the Daemons of Hadoop

    Various daemons of Hadoop are:

    NameNode- It is also known as Master in Hadoop cluster. It stores meta-data i.e. number of Blocks, their location, replicas and other details. NameNode maintains and manages the slave nodes, and assigns tasks to them. It should be deployed on reliable hardware as it is the centerpiece of HDFS.
    Secondary NameNode- It download the FsImage and EditLogs from the NameNode. Then it merges EditLogs with the Fsimage periodically. It keeps edits log size within a limit. Then it store the modified FsImage into persistent storage. So we can use FsImage in case of NameNode failure.
    DataNode- It is also known as Slave node. It is responsible for storing actual data in HDFS. DataNode perform read and write operation as per request for the clients.
    Node Manager- It is the per-machine/per-node framework agent. It is responsible for containers, monitoring their resource usage and reporting the same to the Resource manager.
    ResourceManager- The YARN Resource Manager Service (RM) is the central controlling authority for resource management and makes allocation decisions. ResourceManager has two main components: Scheduler and ApplicationsManager.

18. What are 4 configuration files in Hadoop
    https://www.edureka.co/blog/hadoop-cluster-configuration-files/

    • hadoop-env.sh
    • core-site.xml
    • hdfs-site.xml
    • mapred-site.xml
    • masters
    • slaves

    HADOOP_HOME directory (the extracted directory(etc) is called as HADOOP_HOME. e.g. hadoop-2.6.0-cdh5.5.1) contain all the libraries, scripts, configuration files, etc.

hadoop-env.sh

1. This file specifies environment variables that affect the JDK used by Hadoop Daemon (bin/hadoop).
As Hadoop framework is written in Java and uses Java Runtime environment, one of the important environment variables for Hadoop daemon is $JAVA_HOME in hadoop-env.sh.

2. This variable directs Hadoop daemon to the Java path in the system
Actual:export JAVA_HOME=<path-to-the-root-of-your-Java-installation>
Change:export JAVA_HOME=</usr/lib/jvm/java-8-oracle/>
core-site.sh
3. This file informs Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/dataflair/Hadmin</value>
</property>
```

 Location of namenode is specified by fs.defaultFS property
 namenode running at 9000 port on localhost.
 hadoop.tmp.dir property to specify the location where temporary as well as permanent data of Hadoop will be stored.
 "/home/dataflair/hadmin" is my location; here you need to specify a location where you have Read Write privileges.

hdfs-site.sh
 we need to make changes in Hadoop configuration file hdfs-site.xml (which is located in HADOOP_HOME/etc/hadoop) by executing the below command:
Hdata@ubuntu:~/hadoop-2.6.0-cdh5.5.1/etc/hadoop$ nano hdfs-site.xml

Replication factor

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
```

 Replication factor is specified by dfs.replication property;
 as it is a single node cluster hence we will set replication to 1.

mapred-site.xml
 we need to make changes in Hadoop configuration file mapred-site.xml (which is located in HADOOP_HOME/etc/hadoop)
 Note: In order to edit mapred-site.xml file we need to first create a copy of file mapred-site.xml.template. A copy of this file can be created using the following command:
Hdata@ubuntu:~/ hadoop-2.6.0-cdh5.5.1/etc/hadoop$ cp mapred-site.xml.template mapred-site.xml
 We will now edit the mapred-site.xml file by using the following command:
Hdata@ubuntu:~/hadoop-2.6.0-cdh5.5.1/etc/hadoop$ nano mapred-site.xml
Changes

```
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

In order to specify which framework should be used for MapReduce, we use mapreduce.framework.name property, yarn is used here.

yarn-site.xml
Changes

```
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.
shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

 In order to specify auxiliary service need to run with nodemanager“yarn.nodemanager.aux-services” property is used.
 Here Shuffling is used as auxiliary service. And in order to know the class that should be used for shuffling we user “yarn.nodemanager.aux-services.mapreduce.shuffle.class”

19. Commands
    a. copyFromLocal
    b. moveFromLocal
    c. put
    d. get
    e. copyToLocal
    f. moveToLocal
    g. get
    h. put
    i. mkdir
    j. ls
    h. append
    i. setrep
    j. mv
    k. put
    l. rm
    m. fsck
20. What do you know about Speculative Execution

In MapReduce, jobs are broken into tasks and the tasks are run in parallel to make the overall job execution time smaller than it would otherwise be if the tasks ran sequentially. Now among the divided tasks, if one of the tasks take more time than desired, then the overall execution time of job increases.

Tasks may be slow for various reasons:
Including hardware degradation or software misconfiguration, but the causes may be hard to detect since the tasks may be completed successfully, could be after a longer time than expected. Apache Hadoop does not fix or diagnose slow-running tasks. Instead, it tries to detect when a task is running slower than expected and launches another, equivalent task as a backup (the backup task is called as speculative task). This process is called Speculative execution in MapReduce.

Speculative execution in Hadoop does not imply that launching duplicate tasks at the same time so they can race. As this will result in wastage of resources in the cluster. Rather, a speculative task is launched only after a task runs for the significant amount of time and framework detects it running slow as compared to other tasks, running for the same job.

When a task successfully completes, then duplicate tasks that are running are killed since they are no longer needed.

If the speculative task after the original task, then kill the speculative task.
on the other hand, if the speculative task finishes first, then the original one is killed.
Speculative execution in Hadoop is just an optimization, it is not a feature to make jobs run more reliably.

So if, I summarize:
The speed of MapReduce job is dominated by the slowest task. MapReduce first detects slow tasks. Then, run redundant (speculative) tasks. This will optimistically commit before the corresponding stragglers This process is known as speculative execution. Only one copy of a straggler is allowed to be speculated. Whichever copy (among the two copies) of a task commits first, it becomes the definitive copy, and the other copy is killed by the framework.

```
    What is default size of block?
-  64 MB
    what is default replication factor?
-  3
    Shell we Increase the hdfs block size?
-  Yes,Multiples of 64 MB
    Where will store namenode meta data?
-  Master(editlogs,fsimage)
    What are the daemon processes will be there in hdfs?
-  NameNode, DataNode and SecondaryNameNode
    Different modes of Hadoop ?
-  Local, suedo distribution and distribution
   Hdfs command to format namenode?
-  Hadoop namenode -format
   What is namenode  and jobtracker UI port?
-  Namenode UI - 50070 , Jobtracker - 50030

  Whare the main hdfs configuration files?
-  core-site.xml,hdfs-site.xml,mapred-site.xml
    Where we will configure blocksize,replication?
-  hdfs-site.xml
    If we get any errors while loading  hdfs data where we need to check?
-  In logfiles
    What are Hadoop Basic components?
-  HDFS and MapReduce
    Main Features of Hadoop?
-   Horizontal Scalability, Distributed Storage, Fault Tolerance
    How much storage needed for 1TB data with replication 3 ?
-  3TB
   How many blocks will create for 10 GB file with block size 64 MB?
-  160
   what is the command for file checking
-  Hadoop fsck


'MR'
1. In Map Reduce ideally how many mappers should be configured on a slave

   You cannot change the no of Mappers via new Java API, because we are using Job class in
   MapReduce configuration core. In old API(deprecated), we can set no of mappers using
   setNumMapTasks(int n) methods via the JobConf object. Ideally, this is not the best way to
   set/change the no of mappers.
By default, no of mappers are 2 on each slave-node. We can set/change this value using mapreduce
.tasktracker.map.tasks.maximum parameter. You need to set this parameter in mapred-site.xml file
. We should not directly select random value to set the no of mappers.
Ideally for each logical InputSplit, a independent mapper or map dynamic container will get
invoked. If we go with default case, on each particular slave node, Node-manager can run only
two mappers or map dynamic containers parallely irrespective of logical input splits. Initially
two input split are assigned to two map dynamic containers on slave-node1. then the remaining
input split might be in a queue. In some cases, these input splits might got traveled to some
other
slave-node(Let's say SN2) which is having map dynamic container sitting idle. This mapper can
process the traveled input-split on this slave-node (SN2).
Even though if you specified 2 value(No of mappers) in configuration file. Node-manager doesn't
invoke all mappers parallely. This decision is taken care by Resource Manager based on the input
 split(s) available on a particular slave-node. But that slave-node can run maximum 2 map
dynamic container parallely.
Please go through below one, so that you can come to know how many no of maximum mapper we need
to set in order to get optimize solution on a particular slave node.
When you are setting up the cluster, at that time you should decide how many maximum no of
mappers that should be configured/run parallely on all slaves-nodes. Basically, no of mappers
are decided based on the below two factors, that is,
1) No of cores
2) Ram memory
```

```
Lets say we have 10 cores on your system. we can have 10 mappers(One mapper = one core) if go
with one core for each mapper. Each mapper/map dynamic container can run on one core. This case
might not be true in all cases.
Let’s say you have 10 cores on your slave-node, and ram memory is 25GB. Your job need 5GB of
memory, so every map tasks requires 5GB of ram.You will have 5 cores on each slave-node. So that
 we can run maximum 5 mappers parallely. On slave-node, it doesn't have enough memory to run
more than 5 mappers parallely even though we have more no of cores available on slave-node. In
this case, maximum no of mappers are limited by amount of ram available in your systems. It is
not limited by cores available in your system.
If your job required ,every map tasks to be loaded with 5GB of memory, then you are wasting
cores if you are having 10 cores on each slave-nodes. Here we are using only 5 cores on each
slave-node, remaining 5 cores are not utilized. Either go with "10 cores with 50GB memory" or
"5 cores with 25GB of ram memory". This will gives the optimal usage of resources.
In general, for each mapper, we will go with 1 to 1.5 core processor. If the usage/processing
is very small/light, then go with 1 core processor for each map dynamic container. If the
usage/processing is very heavy, then go with 1.5 core processor for each map dynamic container.
And also you should the keep above two factors in mind to serve the optimized solution.
'




2. How to set no of Mappers in Map Reduce

    n a Single word, no we cannot change the number of Mappers in MapReduce job but we can
    configure Reducers as per our requirement.

    And the number of Mappers depends upon the number of InputSplit and this InputSplit depends
    on Size of your files and Block size. For example- If we have the block size of 128MB, then
    the number of mappers will be approximately 4.

3. Where is output of Mappers Stored

    disk

    The output of the maps jobs is stored in the local disk of the mappers. Once the map job
    finishes these local outputs are then transferred to reducers. You can check your $
    HADOOP_HOME/conf/mapred-site.xml to check where mapper outputs are stored.

4. What is Partitioner and Combiner

    A partitioner divides the intermediate key-value pairs produced by map tasks into partition.
     The total number of partition is equal to the number of reducers where each partition is
    processed by the corresponding reducer. The partitioning is done using the hash function
    based on a single key or group of keys. The default partitioner available in Hadoop is
    HashPartitioner.

    In Hadoop MapReduce concept, we have a class in between Mapper and Reducer, called Combiner.
When a MapReduce(MR) job is run on a large dataset, Map task generates huge chunks of
intermediate data, which is passed on to Reduce task. During this phase, the output from Mapper
has to travel over the network to the node where Reducer is running. This data movement may
cause network congestion if the data is huge.
To reduce this network congestion, MR framework provides a function called 'Combiner', which is
also called as 'Mini-Reducer'
The role of Combiner is to take the output of Mapper as it's input, process it and sends its
output to the reducer. Combiner reads each key-value pair, combines all the values for the same
key, and sends this as input to the reducer, which reduces the data movement in the network.
Combiner works along with each Mapper.
Combiner uses same class as Reducer.
'

5. Explain shuffling and sorting

    Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce
    phase will run here hence called as reducer node). The shuffling is the physical movement of
     the data which is done over the network. Once all the mappers are finished and their output
```

is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase. Follow this comprehensive guide to read more about Shuffling and Sorting in Hadoop MapReduce.

6. Explain input split

It is created by InputFormat, logically represent the data which will be processed by an individual Mapper (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper. Learn MapReduce InputSplit in detail.

7. Explain Record Reader

It communicates with the InputSplit in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

8. Explain Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Follow this link to learn about Reducer in detail.

9. Is map only job possible

For Example Sqoop Job

In Hadoop, Map-Only job is the process in which mapper does all task, no task is done by the reducer and mapper's output is the final output. In this tutorial on Map only job in Hadoop MapReduce, we will learn about MapReduce process, the need of map only job in Hadoop, how to set a number of reducers to 0 for Hadoop map only job. We will also learn what are the advantages of Map Only job in Hadoop MapReduce, processing in Hadoop without reducer along with MapReduce example with no reducer. Learn how to install Hadoop 2 with Yarn on pseudo distributed mode

MapReduce is a software framework for easily writing applications that process the vast amount of structured and unstructured data stored in the Hadoop Distributed Filesystem (HDFS). Two important tasks done by MapReduce algorithm are: Map task and Reduce task. Hadoop Map phase takes a set of data and converts it into another set of data, where individual element are broken down into tuples (key/value pairs). Hadoop Reduce phase takes the output from the map as input and combines those data tuples based on the key and accordingly modifies the value of the key.

From the above word-count example, we can say that there are two sets of parallel process, map and reduce; in map process, the first input is split to distribute the work among all the map nodes as shown in a figure, and then each word is identified and mapped to the number 1. Thus the pairs called tuples (key-value) pairs. In the first mapper node three words lion, tiger, and river are passed. Thus the output of the node will be three key-value pairs with three different keys and value set to 1 and the same process repeated for all nodes. These tuples are then passed to the reducer nodes and partitioner comes into action. It carries out shuffling so that all tuples with the same key are sent to the same node. Thus, in reduce process basically what happens is an aggregation of values or rather an operation on values that share the same key. Now, let us consider a scenario where we just need to perform the operation and no aggregation required, in such case, we will prefer 'Map-Only job' in Hadoop. In Hadoop Map-Only job, the map does all task with its InputSplit and no job is done by the reducer. Here map output is the final output. Refer this guide to learn Hadoop features and design principles.

Advantages of Map only job in Hadoop In between map and reduces phases there is key, sort and shuffle phase. Sort and shuffle are responsible for sorting the keys in ascending order and then grouping values based on same keys. This phase is very expensive and if reduce phase is not required we should avoid it, as avoiding reduce phase would eliminate sort and shuffle phase as well. This also saves network congestion as in shuffling, an output of mapper travels to reducer and when data size is huge, large data needs to travel to the reducer. Learn more about shuffling and sorting in Hadoop MapReduce. The output of mapper is written to local disk before sending to reducer but in map only job, this output is directly written to HDFS. This further saves time and reduces cost as well. Also, there is no need of partitioner and combiner in Hadoop Map Only job that makes the process fast.

10. Explain Distrubuted Cache

In Hadoop, data chunks process independently in parallel among DataNodes, using a program written by the user. If we want to access some files from all the DataNodes, then we will put that file to Distributed Cache.

MapReduce framework provides a service called Distributed Cache to caches files needed by the applications. It can cache read-only text files, archives, jar files etc.

First of all, an application which need to use distributed cache to distribute a file:

Should make sure that the file is available.
And also make sure that file can accessed via urls. Urls can be either hdfs: // or http:// or it can be file://
Now, if the file is present on the mentioned urls. The user mentions it to be a cache file to the distributed cache. MapReduce job will copy the cache file on all the nodes before starting of tasks on those nodes.

Process as Follows:

Copy the requisite file to the HDFS: $ hdfs dfs put/user/dataflair/lib/jar_file.jar
Setup the application's JobConf: DistributedCache.addFileToClasspath(new Path (“/user/dataflair/lib/jar-file.jar”), conf).
Add it in Driver class.

11. Write a word count problem in Map reduce

```
package co.edureka.mapreduce;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;

public class WordCount
{
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {
public void map(LongWritable key, Text value,Context context) throws IOException,
InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

```
}
}

public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException,
InterruptedException {
int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}

public static void main(String[] args) throws Exception {

Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
'

'KAFKA' https://mindmajix.com/apache-kafka-interview-questions
-------
https://data-flair.training/blogs/kafka-interview-questions/



1. Explain Different components of KAFKA

    Kafka Interview Questions- Components of Kafka

    Topic -
    Kafka Topic is the bunch or a collection of messages.

    Producer -
    In Kafka, Producers issue communications as well as publishes messages to a Kafka topic.

    Consumer -
    Kafka Consumers subscribes to a topic(s) and also reads and processes messages from the
    topic(s).

    Brokers -
    While it comes to manage storage of messages in the topic(s) we use Kafka Brokers.

2. Explain role of offsetin Kafka

    There is a sequential ID number given to the messages in the partitions what we call, an
    offset. So, to identify each message in the partition uniquely, we use these offsets.
```

```
3. Explain consumer group

    The concept of Consumer Groups is exclusive to Apache Kafka. Basically, every Kafka consumer
     group consists of one or more consumers that jointly consume a set of subscribed topics.

4. Explain role of zookeeper

     Apache Kafka is a distributed system is built to use Zookeeper. Although, Zookeeper’s main
     role here is to build coordination between different nodes in a cluster. However, we also
     use Zookeeper to recover from previously committed offset if any node fails because it
     works as periodically commit offset

5. Explain the term of leader and follower in Kafka Environment

    In every partition of Kafka, there is one server which acts as the Leader, and none or more
    servers plays the role as a Followers.

6. Why Replications are important in Kafka

    Because of Replication, we can be sure that published messages are not lost and can be
    consumed in the event of any machine error, program error or frequent software upgrades.

7. Explain Kafka Architecture

    Apache Kafka APIs

    a. Producer API
        In order to publish a stream of records to one or more Kafka topics, the Producer API
        allows an application.

    b. Consumer API
        This API permits an application to subscribe to one or more topics and also to process
        the stream of records produced to them.

    c. Streams API
        Moreover, to act as a stream processor, consuming an input stream from one or more
        topics and producing an output stream to one or more output topics, effectively
        transforming the input streams to output streams, the streams API permits an application.

    d. Connector API
        While it comes to building and running reusable producers or consumers that connect
        Kafka topics to existing applications or data systems, we use the Connector API. For
        example, a connector to a relational database might capture every change to a table.



    a. Kafka Broker
        Basically, to maintain load balance Kafka cluster typically consists of multiple brokers
        . However, these are stateless, hence for maintaining the cluster state they use
        ZooKeeper. Although, one Kafka Broker instance can handle hundreds of thousands of reads
         and writes per second. Whereas, without performance impact, each broker can handle TB
        of messages. In addition, make sure ZooKeeper performs Kafka broker leader election.


    b. Kafka – ZooKeeper
        For the purpose of managing and coordinating, Kafka broker uses ZooKeeper. Also, uses it
         to notify producer and consumer about the presence of any new broker in the Kafka
        system or failure of the broker in the Kafka system. As soon as Zookeeper send the
        notification regarding presence or failure of the broker then producer and consumer,
        take the decision and starts coordinating their task with some other broker.

    c. Kafka Producers
        Further, Producers in Kafka push data to brokers. Also, all the producers search it and
        automatically sends a message to that new broker, exactly when the new broker starts.
        However, keep in mind that producer sends messages as fast as the broker can handle, it
        doesn’t wait for acknowledgments from the broker.
```

d. Kafka Consumers
Basically, by using partition offset the Kafka Consumer maintains that how many messages have been consumed because Kafka brokers are stateless. Moreover, you can assure that the consumer has consumed all prior messages once the consumer acknowledges a particular message offset. Also, in order to have a buffer of bytes ready to consume, the consumer issues an asynchronous pull request to the broker. Then simply by supplying an offset value, consumers can rewind or skip to any point in a partition. In addition, ZooKeeper notifies Consumer offset value.

4. Kafka Fundamental Concepts

Here, we are listing some of the fundamental concepts of Kafka Architecture that you must know:

a. Kafka Topics
The topic is a logical channel to which producers publish message and from which the consumers receive messages.

A topic defines the stream of a particular type/classification of data, in Kafka.
Moreover, here messages are structured or organized. A particular type of messages is published on a particular topic.
Basically, at first, a producer writes its messages to the topics. Then consumers read those messages from topics.
In a Kafka cluster, a topic is identified by its name and must be unique.
There can be any number of topics, there is no limitation.
We can not change or update data, as soon as it gets published.

b. Partitions in Kafka
In a Kafka cluster, Topics are split into Partitions and also replicated across brokers.

However, to which partition a published message will be written, there is no guarantee about that.
Also, we can add a key to a message. Basically, we will get ensured that all these messages (with the same key) will end up in the same partition if a producer publishes a message with a key. Due to this feature, Kafka offers message sequencing guarantee.
Though, unless a key is added to it, data is written to partitions randomly.
Moreover, in one partition, messages are stored in the sequenced fashion.
In a partition, each message is assigned an incremental id, also called offset.
However, only within the partition, these offsets are meaningful. Moreover, in a topic, it does not have any value across partitions.
There can be any number of Partitions, there is no limitation.

c. Topic Replication Factor in Kafka
While designing a Kafka system, it's always a wise decision to factor in topic replication. As a result, its topics' replicas from another broker can solve the crisis, if a broker goes down. For example, we have 3 brokers and 3 topics. Broker1 has Topic 1 and Partition 0, its replica is in Broker2, so on and so forth. It has got a replication factor of 2; it means it will have one additional copy other than the primary one. Below is the image of Topic Replication Factor:

Some key points -

Replication takes place in the partition level only.
For a given partition, only one broker can be a leader, at a time. Meanwhile, other brokers will have in-sync replica; what we call ISR.
It is not possible to have the number of replication factor more than the number of available brokers.

d. Consumer Group

It can have multiple consumer process/instance running.
Basically, one consumer group will have one unique group-id.
Moreover, exactly one consumer instance reads the data from one partition in one consumer group, at the time of reading.
Since, there is more than one consumer group, in that case, one instance from each of

```
        these groups can read from one single partition.
        However, there will be some inactive consumers, if the number of consumers exceeds the
        number of partitions. Let’s understand it with an example if there are 8 consumers and 6
         partitions in a single consumer group, that means there will be 2 inactive consumers.
        Read Apache Kafka + Spark Streaming Integration

        So, this was all about Apache Kafka Architecture. Hope you like our explanation.


8. Explain Partitioning Key

    Kafka topics are divided into a number of partitions, which contains messages in an
    unchangeable sequence. Each message in a partition is assigned and identified by its unique
    offset. A topic can also have multiple partition logs like the click-topic has in the image
    to the right. This allows for multiple consumers to read from a topic in parallel.

In Kafka, replication is implemented at the partition level. The redundant unit of a topic
partition is called a replica. Each partition usually has one or more replicas meaning that
partitions contain messages that are replicated over a few Kafka brokers in the cluster. As we
can see in the pictures - the click-topic is replicated to Kafka node 2 and Kafka node 3.

Apache Kafka Partition
It's possible for the producer to attach a key to the messages and tell which partition the
message should go to. All messages with the same key will arrive at the same partition.'


9. Advantages of Kafka

    High-throughput
    We do not need any large hardware in Kafka, because it is capable of handling high-velocity
    and high-volume data. Moreover, it can also support message throughput of thousands of
    messages per second.

    Low Latency
    Kafka can easily handle these messages with the very low latency of the range of
    milliseconds, demanded by most of the new use cases.

    Fault-Tolerant
    Kafka is resistant to node/machine failure within a cluster.

    Durability
    As Kafka supports messages replication, so,  messages are never lost. It is one of the
    reasons behind durability.

    Scalability
    Kafka can be scaled-out, without incurring any downtime on the fly by adding additional
    nodes.

10. Explain (Same as 1)
    a. Producer
    b. Consumer
    c. Broker
    d. topic
    e. partition
11. Main components where the data is processed seamlessly in kakka

12. Difference between Kafka and flume and Why Kafka is better than flume

    Flume and Kakfa both can act as the event backbone for real-time event processing. Some
    features are overlapping between the two and there are some confusions about what should be
    used in what use cases. This post tries to elaborate on the pros and cons of both products
    and the use cases that they fit the best.

    Flume and Kafka are actually two quite different products. Kafka is a general purpose
    publish-subscribe model messaging system, which offers strong durability, scalability and
    fault-tolerance support. It is not specifically designed for Hadoop. Hadoop ecosystem is
```

just be one of its possible consumers.

Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of data from many different sources to a centralized data store, such as HDFS or HBase. It is more tightly integrated with Hadoop ecosystem. For example, the flume HDFS sink integrates with the HDFS security very well. So its common use case is to act as a data pipeline to ingest data into Hadoop.

Kafka is very scalable. One of the key benefits of Kafka is that it is very easy to add large number of consumers without affecting performance and without down time. That's because Kafka does not track which messages in the topic have been consumed by consumers. It simply keeps all messages in the topic within a configurable period. It is the consumers' responsibility to do the tracking through offset. In contrast, adding more consumers to Flume means changing the topology of Flume pipeline design, replicating the channel to deliver the messages to a new sink. It is not really a scalable solution when you have huge number of consumers. Also since the flume topology needs to be changed, it requires some down time.

Kafka's scalability is also demonstrated by its ability to handle spike of the events. This is where Kakfa truly shines because it acts as a "shock absorber" between the producers and consumers. Kafka can handle events at 100k+ per second rate coming from producers. Because Kafka consumers pull data from the topic, different consumers can consume the messages at different pace. Kafka also supports different consumption model. You can have one consumer processing the messages at real-time and another consumer processing the messages in batch mode. On the contrary, Flume sink supports push model. When event producers suddenly generate a flood of messages, even though flume channel somewhat acts as a buffer between source and sink, the sink endpoints might still be overwhelmed by the write operations.

Message durability is also an important consideration. Flume supports both ephemeral memory-based channel and durable file-based channel. Even when you use a durable file-based channel, any event stored in a channel not yet written to a sink will be unavailable until the agent is recovered. Moreoever, the file-based channel does not replicate event data to a different node. It totally depends on the durability of the storage it writes upon. If message durability is crucial, it is recommended to use SAN or RAID. Kafka supports both synchronous and asynchronous replication based on your durability requirement and it uses commodity hard drive.

Flume does have some features that makes it attractive to be a data ingestion and simple event processing framework. The key benefit of Flume is that it supports many built-in sources and sinks, which you can use out of box. If you use Kafka, most likely you have to write your own producer and consumer. Of course, as Kakfa becomes more and more popular, other frameworks are constantly adding integration support for Kafka. For example, Apache Storm added Kafka Spout in release 0.9.2, allowing Storm topology to consume data from Kafka 0.8.x directly.

Kafka does not provider native support for message processing. So mostly likely it needs to integrate with other event processing frameworks such as Apache Storm to complete the job. In contrast, Flume supports different data flow models and interceptors chaining, which makes event filtering and transforming very easy. For example, you can filter out messages that you are not interested in the pipeline first before sending it through the network for obvious performance reason. However, It is not suitable for complex event processing, which I will address in a future post.

The good news is that the latest trend is to use both together to get the best of both worlds. For example, Flume in CDH 5.2 starts to accept data from Kafka via the KafkaSource and push to Kafka using the KafkaSink. Also CDH 5.3 (the latest release) adds Kafka Channel support, which addresses the event durability issue mentioned above.

'

14. ISR in Kafka

Basically, a list of nodes that replicate the log is Replicas. Especially, for a particular partition. However, they are irrespective of whether they play the role of the Leader.

In addition, ISR refers to In-Sync Replicas. On defining ISR, it is a set of message replicas that are synced to the leaders.

```
15. Key advantages of Kafka

Advantages of Kafka
Apache Kafka is selected for it's strengths in the space of messaging. The following are some
of the advantages which Kafka possess, making it ideal for our Data Lake implementation:

High-throughput: Kafka is capable of handling high-velocity and high-volume data using not so
large hardware. It is capable of supporting message throughput of thousands of messages per
second.
Low latency: Kafka is able to handle these messages with very low latency of the range of
milliseconds, demanded by most of new use cases.
Fault tolerant: The inherent capability of Kafka to be resistant to node/machine failure within
a cluster.
Durability: The data/messages are persistent on disk, making it durable and messages are also
replicated ...
'

16. How to create a topic in kafka

./kafka-topics.sh --create --zookeeper localhost:2182 --partitions 2  --replication-factor 1
--topic test_20180613

17. how to start zookeeper

    in/zookeeper-server-start.sh config/zookeeper.properties

    Next, to start the Kafka server: > bin/kafka-server-start.sh config/server.properties

18. What is default retension period of Kafka Broker

    160 Hours

19. How do intergrate Spark Streaming with Kafka









20. How to make RDBMS or Producer
    and RDBMS as consumer









'PIG'
-----

1. Difference between PIG and Hive

Language        Pig Latin               SQL-like
Application     Programming purposes    Report creation
Operation       Client Side             Server side
Data support    Semi-structured         Structured
Connectivity    Can be called
                by other applications   JDBC & BI tool integration
```

```
2. Explain ( ILLUSTRATE,DESCRIBE,EXPLAIN,Define)

DUMP : It helps to display the results on screen.
DESCRIBE : It helps to display the schema of aparticular relation.
ILLUSTRATE : It helps to display step by step execution of a sequence of pig statements
EXPLAIN : It helps to display the execution plan for Pig Latin statements.



3. What are the Data types avaialble in PIG

    Int
    Long
    Float
    Double
    Char array
    Byte array

    Complex :
    Bag
    Map
    Tuple


4. Explain What are the transformation avaialble in PIG
    a. Distinct
    b. filter
    c. for each
    d. order by
    e. group
    f. cogroup
    g. Join
        join
        left outer Join
        Right outer Join
        Full outer join
        cross
    h. limit
    i. Union
    j. split
5. Explain Data types avaialble in PIG --Same as 3
6. Explain Flatten in PIG

    Sometimes there is data in a tuple or a bag and if we want to remove the level of nesting
    from that data, then Flatten modifier in Pig can be used. Flatten un-nests bags and tuples.
    For tuples, the Flatten operator will substitute the fields of a tuple in place of a tuple,
    whereas un-nesting bags is a little complex because it requires creating new tuples.

7. How do you process below formats using PIG
    a. JSON

        ins_json = LOAD 'PIG_SCRIPTS/ins_json' USING JsonLoader
(
'this:float,
that:float,
insight: (
diff : float,
percentage_diff : float,
normalised_diff : float,
normalised_percentage_diff : float,
zscore_diff : float,
zscore_percentage_diff : float,
normalised_zscore_diff : float,
normalised_zscore_percentage_diff : float,
```

```
percentile_rank : float)'
);

json_insigh = foreach ins_json generate this,that,insight.diff,insight.percentage_diff,insight.
normalised_diff,insight.normalised_percentage_diff,insight.zscore_diff,insight.
zscore_percentage_diff,insight.normalised_zscore_diff,insight.normalised_zscore_percentage_diff,
insight.percentile_rank;

dump json_insigh;

store json_insigh;

    b. CSV

    A = LOAD '/tmp/test.csv' USING PigStorage(',') AS (a:chararray, b:chararray, c:chararray, d:
    chararray, e:chararray);

    DUMP A;


    c. XML

    hdfs dfs -copyFromLocal customers_data.xml PIG_SCRIPTS/customers_data.xml

CUSTOMERS_DATA = load 'PIG_SCRIPTS/customers_data.xml' using org.apache.pig.piggybank.storage.
XMLLoader('customer') as (customer:chararray);

grunt> CUSTOMERS_DATA = load 'PIG_SCRIPTS/customers_data.xml' using org.apache.pig.piggybank.
storage.XMLLoader('customer') as (customer:chararray);
grunt> dump CUSTOMERS_DATA;


8. Scenerios that we can you PIG

MapReduce is a powerful programming model based on the principle parallel processing or
computation of data. Hadoop MapReduce gives the programmers the ability to filter and aggregate
data from HDFS to gain business insights from big data. MapReduce programming can be implemented
 using many conventional programming languages like Java, Python, C etc.

On the other hand, Apache Pig is a platform for analyzing large data sets containing high-level
language for expressing data analysis programs, coupled with infrastructure for evaluating these
 programs. It gives ease of programming to the developers by enabling complex programmatical
challenges to be written in simple data flow sequence and less complex textual language.

Most of the jobs can be run using Pig and Hive but to make use of the advanced application
programming interfaces, developers may look up to MapReduce alternatives. In certain situations
we need MapReduce alternative over Pig like below:

1) When Hadoop developers need definite driver program control then they should make use of
Hadoop MapReduce instead of Pig and Hive.
2) When Hadoop developer needs implementing a custom partitioner they choose MapReduce over Pig
and Hive.
3) If there already exists pre-defined library of Java Mappers or for a job then it is a wise
decision to use Hadoop MapReduce instead of Pig and Hive.
4) Hadoop MapReduce can prove to be a better coding approach over Pig and Hive if the job
requires optimization at a particular stage of processing by making the best use of tricks like
in-mapper combining.
5) If the job has some tricky usage of Distributed cache (replicated join), cross products,
groupings or joins then Hadoop MapReduce is a better programming approach over Pig


9. Explain Tuple ,Bag and Map

    Tuple
    An ordered set of fields is what we call a tuple.
    For Example: (Ankit, 32)
```

```
    Bag
    A collection of tuples is what we call a bag.
    For Example: {(Ankit,32),(Neha,30)}
    Map
    A set of key-value pairs is what we call a Map.
    For Example: [ 'name'#'Ankit', 'age'#32]


10. Is PIG case sensitive

    PIG key-words are case insensitive but all other elements are case sensitive.

11. Explain Architecture of PIG


Pig Latin Scripts

Initially as illustrated in the above image, we submit Pig scripts to the Apache Pig execution
environment which can be written in Pig Latin using built-in operators.

There are three ways to execute the Pig script:

Grunt Shell: This is Pig's interactive shell provided to execute all Pig Scripts.
Script File: Write all the Pig commands in a script file and execute the Pig script file. This
is executed by the Pig Server.
Embedded Script: If some functions are unavailable in built-in operators, we can
programmatically create User Defined Functions to bring that functionalities using other
languages like Java, Python, Ruby, etc. and embed it in Pig Latin Script file. Then, execute
that script file.
Parser

From the above image you can see, after passing through Grunt or Pig Server, Pig Scripts are
passed to the Parser. The Parser does type checking and checks the syntax of the script. The
parser outputs a DAG (directed acyclic graph). DAG represents the Pig Latin statements and
logical operators. The logical operators are represented as the nodes and the data flows are
represented as edges.

Optimizer

Then the DAG is submitted to the optimizer. The Optimizer performs the optimization activities
like split, merge, transform, and reorder operators  etc. This optimizer provides the automatic
optimization feature to Apache Pig. The optimizer basically aims to reduce the amount of data in
 the pipeline at any instance of time while processing the extracted data, and for that it
performs functions like:

PushUpFilter: If there are multiple conditions in the filter and the filter can be split, Pig
splits the conditions and pushes up each condition separately. Selecting these conditions
earlier, helps in reducing the number of records remaining in the pipeline.
PushDownForEachFlatten: Applying flatten, which produces a cross product between a complex type
such as a tuple or a bag and the other fields in the record, as late as possible in the plan.
This keeps the number of records low in the pipeline.
ColumnPruner: Omitting columns that are never used or no longer needed, reducing the size of the
 record. This can be applied after each operator, so that fields can be pruned as aggressively
as possible.
MapKeyPruner: Omitting map keys that are never used, reducing the size of the record.
LimitOptimizer: If the limit operator is immediately applied after a load or sort operator, Pig
converts the load or sort operator into a limit-sensitive implementation, which does not require
 processing the whole data set. Applying the limit earlier, reduces the number of records.
This is just a flavor of the optimization process. Over that it also performs Join, Order By and
 Group By functions.

To shutdown, automatic optimization, you can execute this command:

pig -optimizer_off [opt_rule | all ]
Compiler
```

After the optimization process, the compiler compiles the optimized code into a series of MapReduce jobs. The compiler is the one who is responsible for converting Pig jobs automatically into MapReduce jobs.

Execution engine

Finally, as shown in the figure, these MapReduce jobs are submitted for execution to the execution engine. Then the MapReduce jobs are executed and gives the required result. The result can be displayed on the screen using "DUMP" statement and can be stored in the HDFS using " STORE" statement.

After understanding the Architecture, now in this Apache Pig tutorial, I will explain you the Pig Latins's Data Model.

12. Use Cases of PIG

MapReduce is a powerful programming model based on the principle parallel processing or computation of data. Hadoop MapReduce gives the programmers the ability to filter and aggregate data from HDFS to gain business insights from big data. MapReduce programming can be implemented using many conventional programming languages like Java, Python, C etc.

On the other hand, Apache Pig is a platform for analyzing large data sets containing high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. It gives ease of programming to the developers by enabling complex programmatical challenges to be written in simple data flow sequence and less complex textual language.

Most of the jobs can be run using Pig and Hive but to make use of the advanced application programming interfaces, developers may look up to MapReduce alternatives. In certain situations we need MapReduce alternative over Pig like below:

1) When Hadoop developers need definite driver program control then they should make use of Hadoop MapReduce instead of Pig and Hive.
2) When Hadoop developer needs implementing a custom partitioner they choose MapReduce over Pig and Hive.
3) If there already exists pre-defined library of Java Mappers or for a job then it is a wise decision to use Hadoop MapReduce instead of Pig and Hive.
4) Hadoop MapReduce can prove to be a better coding approach over Pig and Hive if the job requires optimization at a particular stage of processing by making the best use of tricks like in-mapper combining.
5) If the job has some tricky usage of Distributed cache (replicated join), cross products, groupings or joins then Hadoop MapReduce is a better programming approach over Pig

13. How fileds are referenced in PIG when schema is not avaialble

```
    salgrade =  load '/user/cloudera/pig/hr/HR/salgrade' using PigStorage(',');
```

14. What are Different in-built functions avaialble in PIG

AVG
CONCAT
COUNT
COUNT_STAR
DIFF
IsEmpty
MAX
MIN
SIZE
SUM
TOKENIZE
Load/Store Functions
Handling Compression
BinStorage
PigDump
PigStorage
TextLoader

```
Math Functions
ABS
ACOS
ASIN
ATAN
CBRT
CEIL
COS
COSH
EXP
FLOOR
LOG
LOG10
RANDOM
ROUND
SIN
SINH
SQRT
TAN
TANH
String Functions
INDEXOF
LAST_INDEX_OF
LCFIRST
LOWER
REGEX_EXTRACT
REGEX_EXTRACT_ALL
REPLACE
STRSPLIT
SUBSTRING
TRIM
UCFIRST
UPPER
Bag and Tuple Functions
TOBAG
TOP
TOTUPLE

15. Difference between group and cogroup

    Group and Cogroup operators are identical. For readability, GROUP is used in statements
    involving one relation and COGROUP is used in statements involving two or more relations.
    Group operator collects all records with the same key. Cogroup is a combination of group and
     join, it is a generalization of a group instead of collecting records of one input depends
    on a key, it collects records of n inputs based on a key. At a time, we can Cogroup up to
    127 relations.

    cogroup_data = COGROUP emp by DEPTNO, dept by DEPTNO;

    pos = foreach baseball_limit generate name, flatten(position) as position;
    bypos = group pos by position;

16. How to get the metadata

    describe

17. UDFx in Pig
18. How do you create pig script and run

    pig -x mapreduce hdfs://localhost:9000/pig_data/Sample_script.pig

19. How to read and store the data

    emp = load 'emp.txt' using PigStorage(',');
    store emp into 'emp';
```

```
20. How do you store processed data in Hive

dw_data_set = LOAD 'default.customers' USING org.apache.hive.hcatalog.pig.HCatLoader();
final_data_set= foreach dw_data_set generate customer_id,customer_fname;
create table customers2 (customer_id int ,customer_fname string);
STORE final_data_set INTO 'default.customers2' USING org.apache.hive.hcatalog.pig.HCatStorer();



'SQL Questions '

1. What is Different types of SQL Statement
2. What are the different Database objects you know
3. What is View ? Types ? and how it is different from Table
4. What is Materialized view and What are the types of refreshed method
5. Difference between view and MV
6. What is Partition and what are different types of partion can be added to table
7. Explain advantage of Using Partitioning in Oracle
8. Exaplain use of Indexes and Different types of Indexes
9. Difference between B-tree and Bitmap Index
10. What do you mean by local and global index
11. What is Synonym and what are the types of synonoyms
12. What you mean by DB-link
13. What are the Data Dictionary tables avaialble in Oracle
14. What are the Different constraints available in Oracle
15. What is different between Table level and column level constraint
16. Use of Sequences
17. What is the Oracle version that you are currently using
18. Explain
    a. DDL
    b. DML
    c. DRL
    d. DCL
19. What are the pre-defined data types avaialble in oracle
    a. Character
    b. Numberic
    c. Date
    d. What are aggregate function
20. Explain working of
    a.  Co-related sub queries
    b.  group by query
21. Explain Different types of Joins available in Oracle
22. How do you delete duplicates from the table
23. Explain Locking mechanism in oracle
24. Explain Use of Global Temporary table (GTT)
25. Difference between  Rank() and Dense_Rank()
26. Explain Use of RowNumber() and Rowid
27. Practice Hierarchiel queries
28. Use of LISTAGG() Queries -- Practice 3 Queries
29. Difference between RowNumber() and rownum
30. Explain the working for B-tree
31. Difference between Delete, Truncate and Drop
32. Explain ACID properties
33. Explain use of Decode() and case
34. Difference between SGA and PGA
35. Explain Complete flow of
     select * from emp ;
36. Explain complete working of
     update emp set ename='VISHAL' where empno=7900;
37. Explain Merge Operation in Oracle.
38. Explain Current of Operation in Oracle
39. Explain types of Sub-Query in Oracle
40. Explain On Delete null and On delete cascade.
41. Difference between varchar vs varchar2 vs Nvarchar2
42. Explain Pseudo Columns in Oracle
43. Explain Sub-partitioning in Oracle.
```

```
44. Explain
    a. Hard Parse
    b. soft parse
45. Explain with respect to oracle Architecture
    a. Blocks
    b. segments
    c. Extents
    d. Data Files
    e. Tablespace
46. Various Hints in Oracle
47. Page no 148 to 185
48. How do you create table faster in Oracle
49. Basic checks you do to improve performance of query
50. Normalization and its Types.
51. Nth Highest Paid Employee
52. Employees with Maximum salary in Each Department
53. Explain
    a. Union
    b. Union all
    c. Intersection
    d. Minus
54. Difference betweeen user_*, all_* and dba_* data Dictionary objects
55. Explain Difference Keys in Oracle


PL/SQL
-------

1. What is the Use of PL/SQL ? What are the Advantages
2. Write an annonyms blocks to update an Employee
3. What are
    a. Procedure
    b. Functions
    c. Packages
    and what are scenarios that above are used
4. Difference between Functions and Procedure
5. What is context switching
6. What is Bulk collect and Bulk Exception
    And when it is used and what is its significance
7. What is Trigger and what are the different types of triggers
8. What is mutating table error
9. Can we use commit in trigger ? Justify the Answer
10. What is Cursor and its types
11. Explain Parameterized cursor
12. What is Ref-Cursor
13. What are Excpetion ? List pre-defined Exception
14. Explain Raise vs Raise Application Error
15. Use of SQLCODE , SQLERRM
16. How do you find the line no Error in PL/SQL -->DBMS_SQLBACKTRACE
17. Collections in PL/SQL
18. Explain Pragma Autonomous Transaction
19. Use of Pragma Exception_INT
20. Modes of Paramter
    a. In
    b. In-out
    c. out
21. Types of Notations
22. Explain Overloaing Procedurs
23. Explain Dynmaic SQL in PL/SQL
24. How do you perform DDL in PL/SQL
25. Check SQL%ROW_COUNT Usage in PL/SQL
26. What are PL/SQL Datatypes
27. Difference between %ROWTYPE AND %TYPE
    AND Explain both
28. Practice Example
    a. Function
```

```
    b. Procedure
    c. Package
    d. Bulk Collect
    e. Bulk collect with Exception
    f. Collectiosn
    g. Cursor
    h. Excpetion
    g. Autonomous Transaction
    h. Dynamic SQL
    i. IF , IF-ELSE
    J. for loop
29. Check Error logging mechanism in Exception from Steven Feuerstein.
30. DBMS Scheduler Jobs in Oracle
31. Doing Activities Fast , Read more on it

    a. Create table with parallel 32 and nologging
    b. Insert /*+ Append*/
    c. create index with parallel 32 and nologging
    d. Disable any triggers while loading any data into table
    e. Parallel session using Shell script and primary key columns


'Data Warehouse'
---------------------
1. What is Surrogate Key
2. What is Normalization and its types
3. What is SCD ? Type 1 and Type 2 Dimention
4. Explain Star Schema
5. Explain Snowflake Schema
6. Explain
    a. Junk Dimentions
    b. Confimed Dimensions
    c. Denerated Dimensions
7. What is ETL
```