SPARK INTERVIEW QUESTIONS

1. **Spark version?**
   Pyspark 3.0.3

2. **What is Spark Context?**
   - Spark Context is a connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.
   - Only one SparkContext should be active per JVM.
   - You must stop () the active SparkContext before creating a new one.

3. **What is Spark Session?**
   - Spark session is a unified entry point of a spark application from Spark 2.0.
   - It provides a way to interact with various spark's functionality with a lesser number of constructs.
   - Instead of having a spark context, hive context, SQL context, now all of it is encapsulated in a Spark session.

4. **Spark1 Vs Spark2?**
   Spark1:
   1. Permanent tables not created in spark 1 without hive integration but In
   2. we can only create spark context.

   Spark2:
   1. We can create permanent tables without hive integration.
   2. We can create both spark session and spark context.
   3. Spark 2 Data stored in Local file system & Metadata will be stored in embedded derby database.

5. **How is Apache Spark different from MapReduce?**

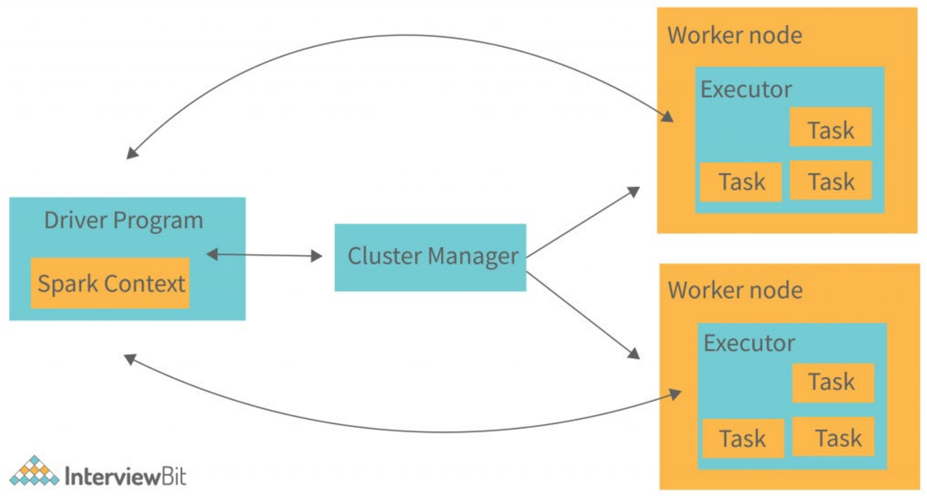
Apache Spark
- Apache Spark is an open-source, distributed processing system used for big data workloads.
- Spark processes data in batches as well as in real-time.
- Spark runs almost 100 times faster than Hadoop MapReduce.
- Spark stores data in the RAM i.e. in-memory. So, it is easier to retrieve it.
- Spark supports in-memory data storage and caching and makes it a low latency computation framework.
- Spark has its own job scheduler due to the in-memory data computation.

MapReduce

- MapReduce processes data in batches only.
- Hadoop MapReduce is slower when it comes to large scale data processing.
- Hadoop MapReduce data is stored in HDFS and hence takes a long time to retrieve the data.
- MapReduce highly depends on disk which makes it to be a high latency framework.
- MapReduce requires an external scheduler for jobs.

6. Spark architecture?

- The Spark follows the master-slave architecture.
- Its cluster consists of a single master and multiple slaves.
- The Spark architecture depends upon two abstractions:
  - ✓ Resilient Distributed Dataset (RDD)
  - ✓ Directed Acyclic Graph (DAG)
- We also have few components in the spark Architecture those are:
  - ✓ Driver node
  - ✓ worker node
  - ✓ Executor
  - ✓ Cluster manager
  - ✓ Task



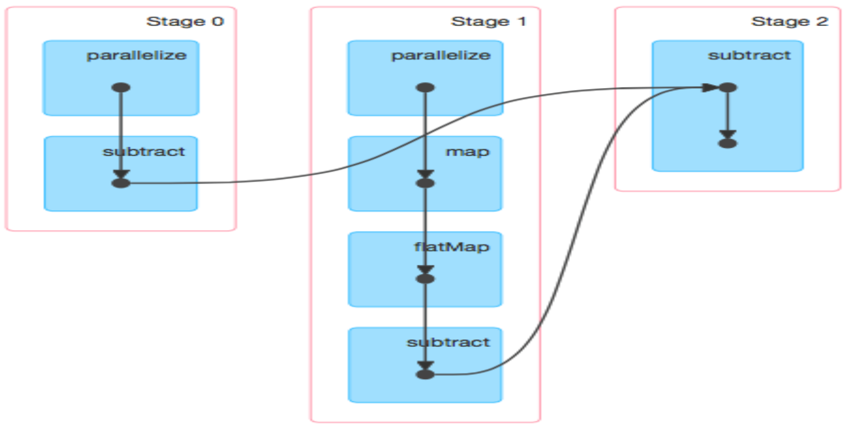
7. What is RDD tell me in brief?

- Introduced in 1X version 2011.
- Resilient Distributed Datasets (RDD)
- **Resilient:** Restore the data on failure.
- **Distributed:** Data is distributed among different nodes.
- **Dataset:** Group of data.
- RDD is the core data object in Spark.

- RDD supports the **Lazy evaluation**.
- RDD's are **Immutable data structures**. Once created, it cannot be modified.
- RDD is designed to be **fault tolerant.**
- It can efficiently process structured data.
- RDD holds the data.
- RDDs are created by either transformation of existing RDDs or by loading an external dataset from stable storage like HDFS or HBase.
- **Type Interface:** RDD provides a uniform interface for processing data from a variety of data sources, such as HDFS, HBase, Cassandra, and others.

8. What is the working of DAG in Spark?

- DAG is a scheduling layer in a spark which implements stage-oriented scheduling.
- It converts logical execution plan to a physical execution plan.
- DAG stands for Direct Acyclic Graph which has a set of finite vertices and edges and stages.
- The vertices represent RDDs, and the edges represent the operations to be performed on RDDs sequentially.
- When an action is called, spark directly strikes the DAG scheduler. It executes the tasks that are submitted to the scheduler.



9. What is driver node, worker node and executor?

1. Driver:

- Logical unit created out of certain amount of memory and certain amount number of cores.
- Is program supply logic to all Data nodes.
- It Creates and Monitor Executors.
- It acts as master in spark application.
- Master scheduled the tasks for the worker nodes.
- The master decides what amount of resources need to be allocated and then based on their availability,

2. **Worker Node:**
    - We have data partition in worker node.
    - A worker node is like a slave node.
    - it gets the work from its master node and executes it.
    - Spark application in a cluster runs in worker nodes.
    - The worker nodes do data processing and reports used resources to the master.

3. **Executor:**
    - Logical unit created out of certain amount of memory and certain amount number of cores.
    - Is a program which processes/computes the partition data with supplied logic by Driver.
    - All executors run under control of the driver.
    - It acts as slave in spark application.

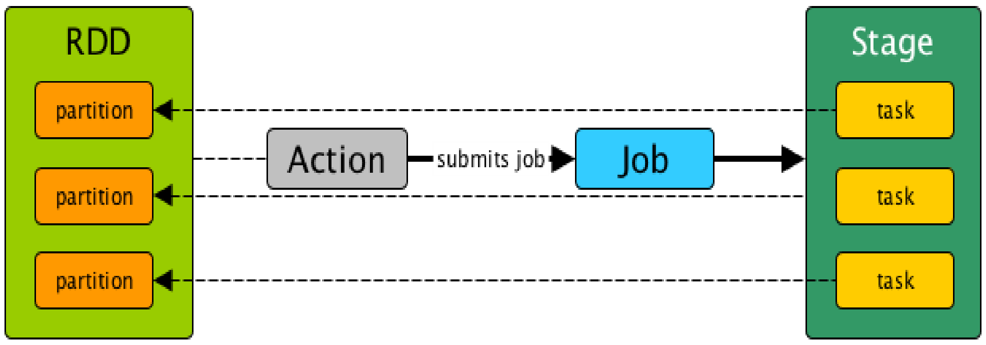
10. **What are stages and tasks in spark?**

Task:

- A unit of work that will be sent to one executor.
- A partition under execution.

Stages:

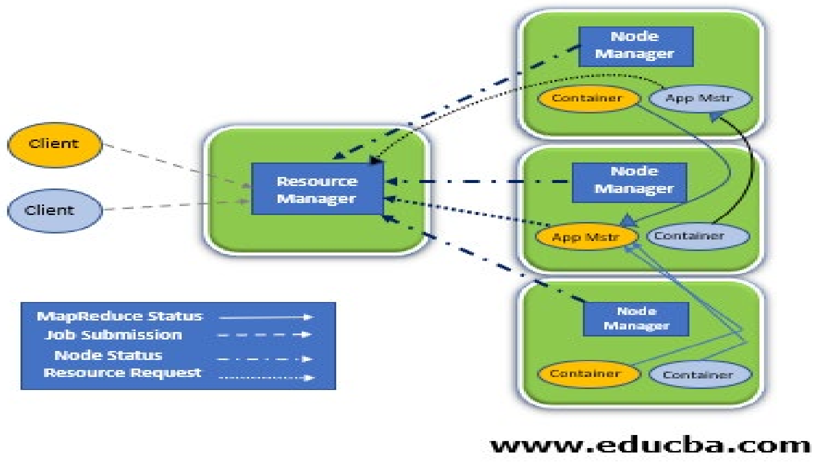
- It is execution of sequence of narrow transformations.
- A New stage will get created when wide transformation occurs.



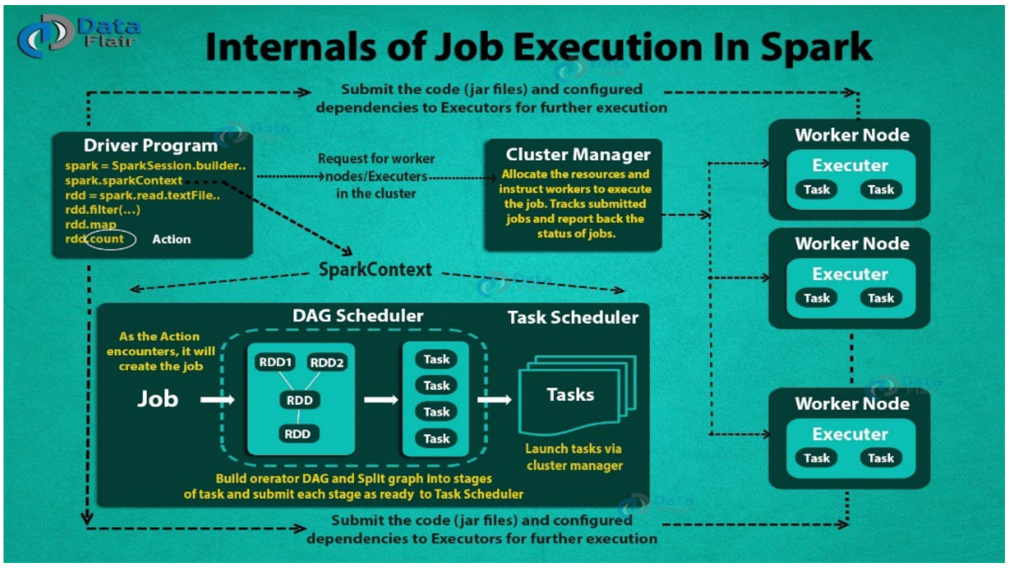
11. What is YARN in Spark?

- YARN is a generic resource-management framework for distributed workloads.
- Cluster-level operating system.
- Master slave architecture.
- Master is Resource Manager and slave is Node manager.

- The resource manager allocates the resources (memory & Processor (cores)) to all nodes in cluster.



12. What happens when we submit spark job /Explain spark run time architecture?


Data Flair
Internals of Job Execution In Spark
Submit the code (jar files) and configured dependencies to Executors for further execution
Driver Program
spark = SparkSession.builder..
spark.sparkContext
rdd = spark.read.textFile..
rdd.filter(...)
rdd.map
rdd.count
Action
Request for worker nodes/Executers in the cluster
Cluster Manager
Allocate the resources and instruct workers to execute the job. Tracks submitted jobs and report back the status of jobs.
Worker Node
Executer
Task
Task
SparkContext
DAG Scheduler
Task Scheduler
As the Action encounters, it will create the job
Job
RDD1
RDD2
RDD
RDD
Task
Tasks
Launch tasks via cluster manager
Build orerator DAG and Split graph into stages of task and submit each stage as ready to Task Scheduler
Submit the code (jar files) and configured dependencies to Executors for further execution

- Spark-Submit is Script, it will submit a spark job to a cluster.
- Spark-submit will launch the Driver which will execute the main () method of our code.
- The driver contacts the cluster manager and requests for resources to launch the Executors.
- The cluster manager launches the Executors on behalf of the Driver.
- It follows master-slave architecture. here driver is master and executors are slave.
- Once the Executors are launched, they establish a direct connection with the Driver.

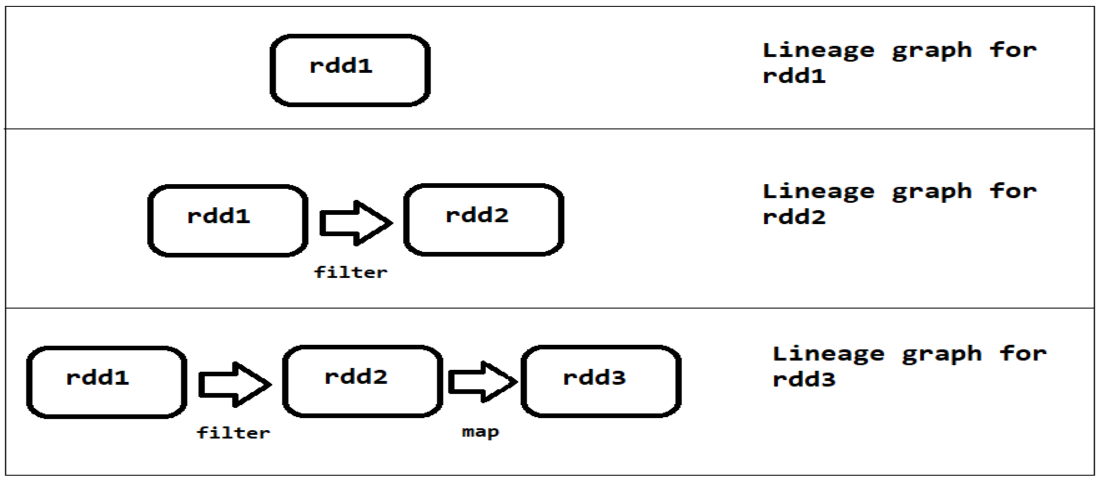
- The driver program converts the code into DAG,which will have all the RDDs and transformations to be performed on them.
- The Driver program converts the DAG to a physical execution plan with a set of stages.
- The driver creates small execution units called tasks.
- The driver determines the total number of Tasks.
- The driver allocates the Tasks to the Executors.
- Task runs on Executor and each Task upon completion returns the result to the Driver.
- Finally, when all Task is completed, the main() method running in the Driver exits, i.e. main() method invokes sparkContext.stop().
- Finally, Spark releases all the resources from the Cluster Manager.

13. Blocks/partitions and tasks and no.of executors?

No.of Blocks/No.of Partitions = No.of input tasks/ output tasks = No.of excutors

14. What is a Lineage Graph?

- A Lineage Graph is a dependencies graph between the existing RDD and the new RDD.
- It means that all the dependencies between the RDD will be recorded in a graph, rather than the original data.
- The need for an RDD lineage graph happens when we want to compute a new RDD or if we want to recover the lost data from the lost persisted RDD.
- Spark does not support data replication in memory. So, if any data is lost, it can be rebuilt using RDD lineage.
- It is also called an RDD operator graph or RDD dependency graph.



15. Define Lazy Evaluation in Spark?

- Lazy Evaluation means that You can apply as many **TRANSFORMATIONS** as you want, but Spark will not start the execution of the process until an **ACTION** is called.
- The benefit of this approach is that Spark can make optimization decisions after it had a chance to look at the DAG in entirety.
- This lazy evaluation increases the system efficiency.

16. What are partitions in PySpark?

PySpark Partition is a way to split a large dataset into smaller datasets based on one or more partition keys. You can also create a partition on multiple columns using partitionBy (), just pass columns you want to partition as an argument to this method.

17. How to check the no of partition of the data frame after partition?

rdd.getNumPartitions()

18. What is the difference between repartition and coalesce?

Repartition

- Use to Increase no.of output partitions.
- Repartition is a wide transformation.
- we can repartition based on column specific to increase the performance.
- Repartition creates new data partitions and performs a full shuffle (physical data movement) of evenly distributed data.
- Repartition internally calls coalesce with shuffle parameter thereby making it slower than coalesce.
- Which is quite an expensive operation.

Coalesce

- Use to decrease no.of output partitions.
- is a Narrow transformation.
- adjust data in existing partition, no shuffling.
- Coalesce is faster than repartition.

19. What do you understand by Shuffling in Spark? When does it occur?

- Data transfer across the network occurs while joining two tables or while performing Key operations such as **GroupByKey, ReduceByKey,Repartition**.
- The process of redistribution of data across different partitions in all nodes.

20. How to optimize code and performance in spark?

1. Filter only required data as early as possible.
2. Avoid UDF's (User Defined Functions).

3. Use DataFrame over RDD:
    - ✓ RDD is used for low level operation with less optimization.
    - ✓ DataFrame is the best choice in most cases due to its catalyst optimizer and low garbage collection (GC) overhead.
4. Use Serialized data formats:
    - ✓ By default, Spark uses the Java serializer on the JVM platform. Instead of Java serializer, Spark can also use another serializer called Kryo. The Kryo serializer gives better performance as compared to the Java serializer.
    - ✓ Kryo serializer is in a compact binary format and offers approximately 10 times faster speed as compared to the Java Serializer.
5. Use Advance Variable: Broadcast and Accumulator.
6. Caching data in memory.
7. Use coalesces () over repartition ()
8. Use mapPartitions() over map()
9. Use reduceByKey over of groupBy
10. Use parquet file with snappy compression which gives the high performance and best analysis.

if properly used above spark optimization, it can.

- Eliminate the long-running job process.
- Improves performance time by managing resources.

21. Define Spark DataFrames?

- Introduced in 1.3X version 2013 to overcome the limitations of the RDD.
- It can efficiently process semi structured & structured data.
- DataFrames are organized into rows and columns.
- DataFrames are like relational database tables.
- Dataframes can read and write the data into various formats like CSV, JSON, AVRO, HDFS, and HIVE tables.
- It is already optimized to process large datasets for most of the pre-processing tasks so that we do not need to write complex functions on our own.
- A Spark DataFrame is an immutable set of objects organized into columns and distributed across nodes in a cluster.
- They allow developers to debug the code during the runtime which was not allowed with the RDDs.
- It uses a catalyst optimizer for optimization purposes. If you want to read more about the catalyst optimizer

22. Define Spark Dataset.

- Introduced in 1.6X version 2016.
- It is an extension of Dataframes with more features like type-safety and object-oriented interface.
- It can efficiently process both structured and unstructured data.
- It also uses a catalyst optimizer for optimization purposes.
- It will also automatically find out the schema of the dataset by using the SQL Engine.
- Dataset is faster than RDDs but a bit slower than Dataframes.
- Users of RDD will find it somewhat similar to code but it is faster than RDDs.
- We cannot create Spark Datasets in Python yet. The dataset API is available only in Scala and Java only.

23. What is Catalyst optimizer?

- To convert data frame programming into RDD programming there are multiple approaches.
- Which approach will enhance application increase the performance of application that decision will be taken care by a programming called **Catalyst Optimizer.**
- This programme internally creates plans for best performance is **called logical plan** and **Physical plan.**

24. Difference between RDD vs DF vs DS?

	DATA FRAME	**DATA SET**	**RDD**
SERIALIZATION	Java serilizer/Kryo serilizer	encoders	Java serilizer/Kryo serilizer
CATALYST OPTIMIZER	Supported	Supported	No
LAZY EVALUATION	Supported	Supported	Supported
PERSISTENCE	Supported	Supported	Supported
FILE EXISTENCE VERIFICATION	Supported	Supported	No
TYPE OF OPERATIONS SUPPORTED	sql	Functional,Sql(from spark 2) (Ds api = RDD api +DF api)	functional
FILES PREFERRED	Structured and semi-structured	Structured, semi-structured and unstructured	unstructured
sechema	yes	Yes	no

25. What is Caching and Persisting?

- Both caching and persisting are used to save the Spark RDD/ Dataframe/and Dataset's.
- Cache () – Always in Memory
- Persist () – Memory and disks
- If we apply RDD.Cache() it will always store the data in memory.
- If we apply RDD.Persist() then some part of data can be stored into the memory some can be stored on the disk.

26. Accumulators in spark?

- Spark Accumulators are shared variables which are only “added” through an associative and commutative operation and are used to perform counters or sum operations.
- Accumulators are variables that are used for aggregating information across the executors.

27. What is broadcast variable?

- Broadcast variables are read-only shared variables that are cached and available on all nodes in a cluster in order to access or use by the tasks.

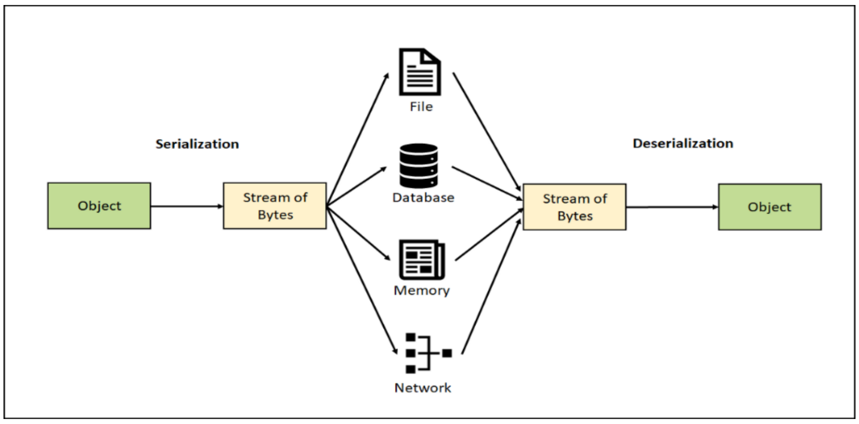
- Instead of sending this data along with every task, spark distributes broadcast variables to the machine using efficient broadcast algorithms to reduce communication costs.

28. Serilization and Deserilization in Spark?

Serialization: is the process of converting a JVM object (RDD/DF/DS) into a stream of bytes.

Deserilization: Construction of JVM objects From Stream of bytes.

When data shuffles from one partition another partition over a network data need to be serialized. To store the the data in partition the data needs to be deserilized.



29. Explain the types of operations supported by RDD?

RDDs support 2 types of operation:

1. **Transformation:** is a function that produces new RDD from the existing RDDs.
   i. Ex**:** Narrow transformations - Map (), flatMap (), filter ()
   ii. Wide Transformations - groupByKey () and reduceByKey()

2. **Actions:** An operation takes input, gives output as a value or collection of values.
   i. Ex: collect (), show (), count (), first (), take(n), top (), countByValue (), reduce ()

30. Types of transformations?

There are fundamentally two types of transformations:

1. **Narrow transformation:**
   - Only one partition data used.
   - No shuffle
   - This Transformation are the result of **map (), mapPartition(), flatMap(), filter(), union()**

2. **Wide Transformations**
   - Multiple partitions data used.
   - Leads to shffle
   - This Transformation is a result of groupByKey **(), aggregateByKey(), aggregate(), join(), repartition().**

31. Difference between group by Key and reduce by Key?

Both reduceByKey and groupByKey result in wide transformations which means both trigger a shuffle operation.

groupByKey:

- group By Key does not do a map side combine.
- creates a lot of shuffling which hampers the performance.

reduceByKey:

- Use reduceByKey as it performs map side combine which reduces the amount of data sent over the network during shuffle.
- It does not shuffle the data as much.

Therefore, reduceByKey is faster as compared to groupByKey.

32. **Difference between Spark Map vs Flat Map Operation?**

i. Spark Map Transformation

- one-to-one transformation. It transforms each element of a collection into one element of the resulting collection.
- Spark Map function takes one element as input process and returns one element at a time.
- Map transforms an RDD of length N into another RDD of length N.
- The input and output RDDs will have the same number of records.

ii. Spark FlatMap Transformation Operation:

- one-to-many transformation. It transforms each element to 0 or more elements
- A FlatMap function takes one element as input returns 0 or more element at a time.
- flatMap() transforms an RDD of length N into another RDD of length M.
- The input and output RDDs will have the same number of records.

33. Difference between map vs map values?

Map ():

- when you just want to transform both key and values use map ().

Mapvalues():

- When you just want to transform the values and keep the keys as-is, it's recommended to use mapValues.

34. List the types of Deploy Modes in Spark.

There are mainly 2 deploy modes in Spark. They are:

Client Mode:

- The default deployment mode is client mode.
- In Client mode when we submit the Spark job/application on Edge node then the Spark driver will run on this edge node.
- In client mode, if a user session terminates, your application also terminates with status fail.
- Client mode is not used for Production jobs. This is used for testing purposes.

Cluster Mode:

- In Cluster Deploy mode, the driver program would be launched on worker nodes.
- Cluster deployment is mostly used for large data sets where the job takes few mins/hrs to complete.

- In this mode, there is a dedicated cluster manager (YARN, Apache Mesos, Kubernetes, etc) for allocating the resources required for the job to run.

35. Fault tolerant in spark?

All the Spark RDDs are immutable. It remembers the dependencies between every RDD involved in the operations, through the lineage graph created in the DAG, and in the event of any failure, Spark refers to the lineage graph to apply the same operations to perform the tasks.

36. Why is Python slower than Scala?

- Python objects are converted into java (JVM) objects using Py4j library vice-versa.
- Scala objects are by default java (JVM) objects.
- Hence No conversion needed, so Scala is faster than python.

37. Spark Default parallelism?

spark will read data from hdfs source as 10 partitions, user can define default parallelism in spark.

df=spark.read.format('csv').load('hdfs://172.16.38.131.8020/bigdata/cse/app_prod/cse.app_prod.csv',10)

38. Spark Input & Output formats?

- By default, spark will read files in snappy. Parquet
- By default, spark will save out files in snappy. Parquet
- Users need to define input and output formats by using format Function.

39. Data skewness?

Some partitions get more data (no.of rows) that will take more time complete, and some partitions get less data (no of rows) takes less time to complete, this will affect overall performance.

Usually, in Apache Spark, data skewness is caused by transformations like join, groupBy, and orderBy, that change data partitions.

40. Salting technique?

In Spark, SALT is a technique to partition data evenly. It's usually good to adapt for wide transformation requires shuffling, like join ,group by operation.

41. **What are the different persistence levels in Apache Spark?**

- **MEMORY_ONLY:** The RDD partitions are stored only in memory.
- **MEMORY_AND_DISK:** if partitions not fitting on the memory will be stored on disk and the data will be read from the disk as and when needed.
- **DISK_ONLY:** The RDD partitions are stored only on the disk.
- **OFF_HEAP:** This level is the same as the MEMORY_ONLY_SER but here the data is stored in the off-heap memory.
- **MEMORY_ONLY_SER:** The RDD is stored as serialized Java Objects as One Byte per partition.
- **MEMORY_AND_DISK_SER:** This level is similar to MEMORY_ONLY_SER but the difference is that the partitions not fitting in the memory are saved on the disk to avoid recomputations on the fly.

42. Spark job running slow in production, how to improve speed?

a. Check resource allocation.
b. Analyze the execution plan.
c. Optimize data partitioning.
d. Tune Spark configurations
e. Use appropriate caching.
f. Enable data compression.
g. Upgrade Spark version.

43. Dynamic Resource Allocation?

✓ Spark provides a mechanism to dynamically adjust the resources based on the workload.
✓ This means that your application may give resources back to the cluster if they are no longer used and request them again later when there is demand.
✓ The values are picked up based on the requirement (size of data, amount of computations needed) and released after use.
✓ This helps the resources to be re-used for other applications.

44. Static Resource Allocation?

The Spark Configuration values are given as part of spark-submit.

45. How to connect hive from spark?

Spark connects directly to the Hive metastore, not through HiveServer2.
To configure this,

- Put hive-site.xml on your classpath, and specify hive.metastore.urls to where your hive metastore hosted

- Import org.apache.spark.sql.hive.HiveContext, as it can perform SQL query over Hive tables

46. How can data transfers be minimized while working with Spark?

Data transfers correspond to the process of shuffling. Minimizing these transfers results in faster and reliable running Spark applications.

- ✓ **Usage of Broadcast Variables**: Broadcast variables increases the efficiency of the join between large and small RDDs.
- ✓ **Usage of Accumulators:** These help to update the variable values parallelly during execution.
- ✓ Another common way is to avoid the operations which trigger these reshuffles.

47. Spark default Executor Memory, Number of Executors, and executor core?

- Default Number of Executors -2
- Default Driver Memory is - 1 GB.
- Default Executor Memory is - 1 GB.
- Default executor cores are - 1 core will be added/allocated-- local/local [1].
- - local[n] - n cores will be added/allocated
- - local [*] - all available cores in cluster

Ex: from pyspark.sql import sparkSession
spark=sparkSession.builder.master(local[*]).appName('demo').getOrCreate()

48. What is the difference between Number of Executors, executor core and Executor Memory in Spark?

Number of Executors: This specifies the number of Executors that are launched on each node in the Spark cluster. By default, this is set to 2, but it can be increased or decreased based on the requirements of the application. This configuration option can be set using the --num-executors flag when launching a Spark application.

Executor Memory: This specifies the amount of memory that is allocated to each Executor. By default, this is set to 1g (1 gigabyte), but it can be increased or decreased based on the requirements of the application. This configuration option can be set using the --executor-memory flag when launching a Spark application.

Executor Cores: This specifies the number of CPU cores that are allocated to each Executor. By default, this is set to 1 core, but it can be increased or decreased based on the requirements of the application. This configuration option can be set using the --executor-cores flag when launching a Spark application.

49. **Distribution of Executors, Cores and Memory for a Spark Application running in Yarn?**

Cluster Config:
10 Nodes
16 cores per Node
64GB RAM per Node

BALANCE APPROACH FOR ANY SIZED CLUSTER:

- **LEAVE 1 CORE PER NODE for Hadoop/Yarn daemons => Num cores available per node = 16-1 = 15**
- TOTAL CORES IN CLUTER = no.of cores per node * no.of nodes in cluster . so, total available of cores in cluster = 15 x 10 = **150**
- **LET'S ASSIGN 5 CORES PER ONE EXECUTOR= 5 (for good hdfs throughput)**
- NUMBER OF AVAILABLE EXECUTORS IN CLUTER = (total cores/num-cores-per-executor) ,so number of available executors = 150/5 = **30**
- NUMBER OF EXECUTORS PER ONE NODE = (number of available executors/ no.of nodes in cluster),so number of executors per one node = 30/10 = **3**
- MEMORY PER ONE EXECUTOR = (available memory in one node / number of executors per one node).so,memory per executor = 64gb/3 = **21GB**
- **Counting off heap overhead = 7% of 21GB = 3GB.**
- **ACTUAL EXECUTOR-MEMORY = 21 - 3 = 18GB**
- **Leaving 1 EXECUTOR for Application Manager .**
- **Then NO.OF EXECUTORS = 29**

So, recommended config is: 29 EXECUTORS, 18GB MEMORY EACH AND 5 CORES EACH!!

50. **Out-of-Memory Errors (OOM)?**

Executor OOM:

- This Error occurs when an executor runs out of memory while processing data.
- It could happen if the data processed or cached in memory is larger than the available executor memory.

Driver OOM:

The Spark driver runs the main program and holds the metadata of the application. If the driver's memory is not properly configured, it can lead to driver OOM errors.