Spark

1. **Spark version?**
   Pyspark 3.0.3

2. **Cluster size?**
   Daily Data:-
   1024 GB per Day
   Number of Core in each node:-
   24 nodes per node 40 TB HDD
   24 Cores per machine ,can around 30 tasks at a time
   Memory (RAM) size:-
   1 GB per task for 30 tasks 30GB
   OS and other related activities – which could be around 3-4GB
   each node will have 30 GB + 4 GB = 34 GB RAM.
   Network Configuration:-
   100 GB/sec at least

3. **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.

4. **Master and slave in different clusters?**

Types of Clusters

1.yarn cluster : Distributed computing cluster (hadoop)
2.Standalone :
3.mesos :
4.kubernetes-Docker :

Yarn Cluster ---> HDFS + Yarn :

Hadoop (HDFS + yarn) --> Master slave architecture

	Master	**Slave**
	----------	**----------**
HDFS -->	Name Node ,Secondary Name Node	DataNode
Yarn--->	Resource Manager	Node Manager

5 Node Yarn cluster

---

n1(master Node)	**n2(slave Node)**	**n3(slave Node)**	**N4(slave Node)**	**n5(slave Node)**
Name Node	DataNode	DataNode	DataNode	DataNode
Secondary Name Node, Resource Manager	Node Manager	Node Manager	Node Manager	Node Manager

Ports Numbers:

Node	**Http port number**	**Rpc port Number**
Name Node	**9870**	**8020/9000**
Resource Manager	**8088**	**8032**

Yarn: When we submit a Spark application or job in yarn cluster

1. partitions are created in Data node (slave Node/worker Node).
2. Node manager manages resources for Data node (slave/worker Node).
3. Resource manager allocate the resources (memory & Processor (cores)) to all nodes in cluster.

5. **What are driver and executor?**

When we submit a spark application or job two types of programs will be created And Resources (ram & cores) Must be allocated by Resource manager.

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 where we have partition.
    - It Creates and Monitor Executors.
    - It acts as master in spark application.
2. **Executor:**
    - Logical unit created out of certain amount of memory and certain amount number of cores.
    - Is program which processes/compute the partition data with supplied logic by Driver.
    - All executors run under control of driver.
    - It acts as slave in spark application.

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

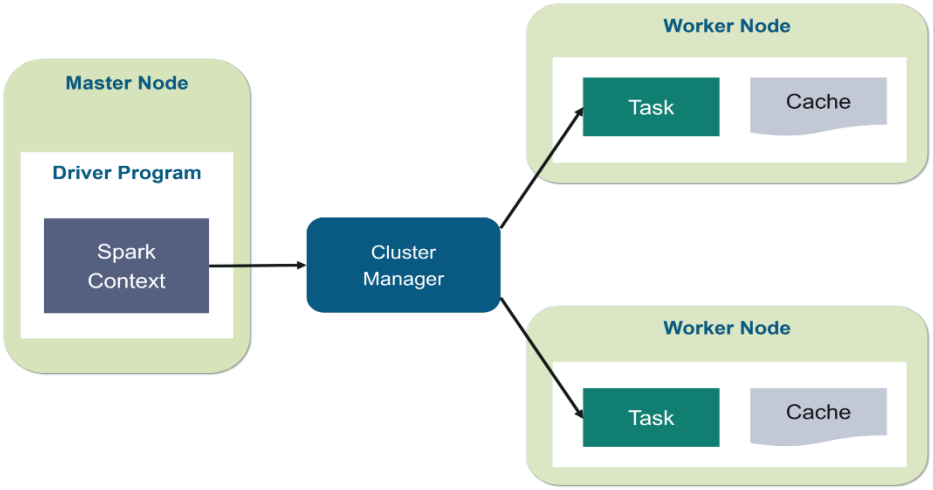
- **Spark-Submit**: Script/Command to submit a spark application to a cluster (Irrespective of cluster, language)
- **Spark Application**: when we submit a spark Application 1 Driver and Multiple Executors per Application are Created.
- Partitioned Data will be on stored in worker Node.
- Meta Data stored in Name Node.

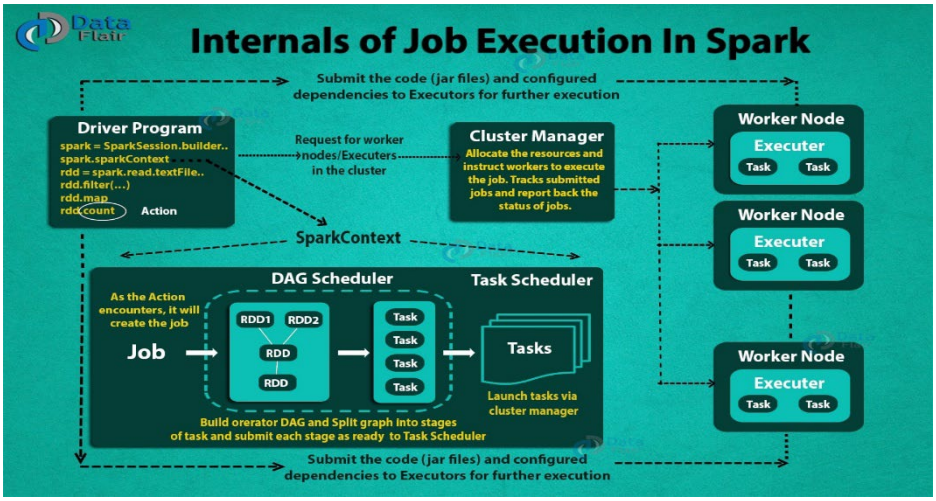
5 Node Yarn-Spark cluster

M1(master Node)	w1(worker Node)	w2 (worker Node)	w3(worker Node)	W4(worker Node)
Name Node	DataNode	DataNode	DataNode	DataNode
Secondary Name Node, Resource Manager	Node Manager	Node Manager	Node Manager	Node Manager

- **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 Directed Acyclic Graph(DAG) which will have all the RDDs and transformations to be performed on them.
- driver program converts the DAG to a physical execution plan with set of stages.
- After this physical plan, driver creates small execution units called tasks.
- The driver determines the total number of Tasks.
- Once the Physical Plan is generated, Spark 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.





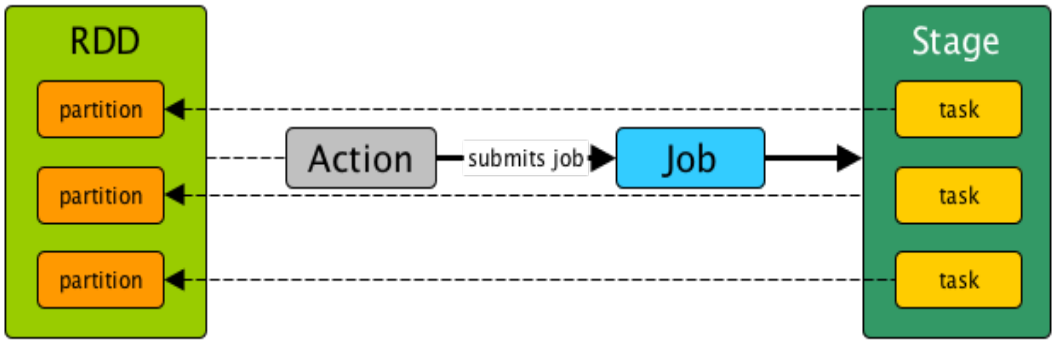
7. Blocks/partitions and tasks and no.of executors?

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

8. **What are stages and tasks in spark?**
   Task: A partition under execution.

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

   We can uniquely identify a stage with the help of its id. Whenever it creates a stage, DAGScheduler increments internal counter nextstageId. It helps to track the number of stage submissions.



9. **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.

   However, if there are unequal-sized data partitions, the speed might be slightly slower.

10. **What is RDD tell me in brief?**

- Introduced in 1X version 2011.
- RDDs or Resilient Distributed Datasets is the core data object in the Spark.
- It can efficiently process structured data.
- RDD holds the data.
- For parallel processing data is partitioned across the distributed cluster of nodes.
- RDDs are created by either transformation of existing RDDs or by loading an external dataset from stable storage like HDFS or HBase.
- RDD is supports the **Lazy evaluation**.
- RDD's are **Immutable data structures**. Once created, it cannot be modified.
- RDD is designed to be **fault-tolerant.**
- **Type Interface:** RDD provides a uniform interface for processing data from a variety of data sources, such as HDFS, HBase, Cassandra, and others.

11. Define Spark DataFrames.

- Introduced in 1.3X version 2013 to overcome the limitations of the Spark RDD.
- It can efficiently process semi structured & structured data.
- 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.
- 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.
- It uses a catalyst optimizer for optimization purposes. If you want to read more about the catalyst optimizer
- DataFrames are a SparkSQL data abstraction and are similar to relational database tables or Python Pandas DataFrames.

12. 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

13. 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

14. 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.
- For example, this information is a API diagnosis like how many records are corrupted or how many times a particular library API was called.

15. **What are Broadcast Variables?Why do we need broadcast variables in Spark?**
Broadcast variables :

- 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.

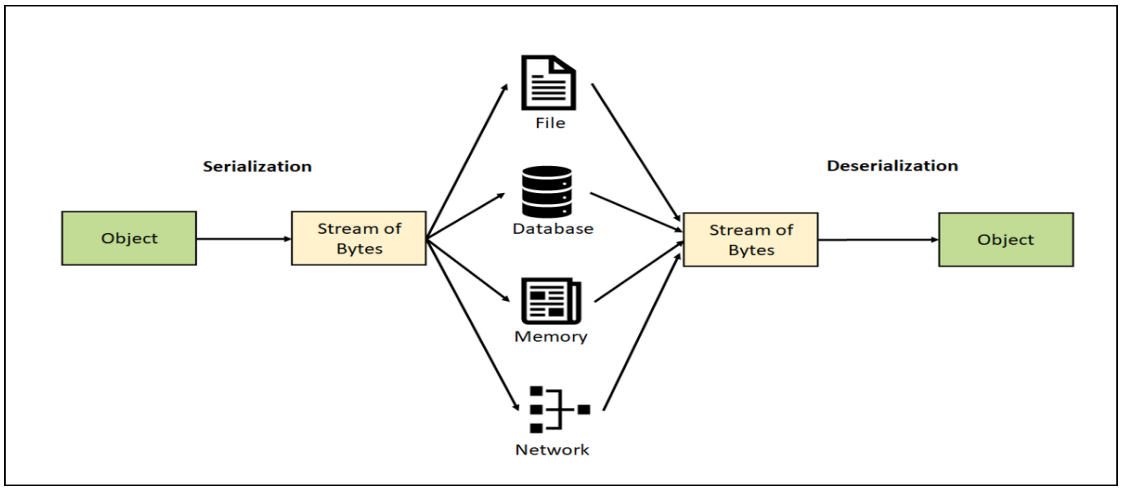
```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)
scala> broadcastVar.value
res0: Array[Int] = Array(1, 2, 3)
```

16. 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 shuffle from one partition another partition over a network data need to be serialized. To store the the data in partition the data need to be deserilized.



17. Explain the types of operations supported by RDDs.

RDDs support 2 types of operation:

Transformations: An operation takes input (data object) gives output as (data object).There are fundamentally two types of transformations:

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

2. **Wide Transformations**

- Multiple partitions data used
- Leads to shffle
- This Transformation is a result of **groupbyKey(), reducebyKey(**).

18. Difference between groupbyKey and reducebyKey?

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

groupByKey:

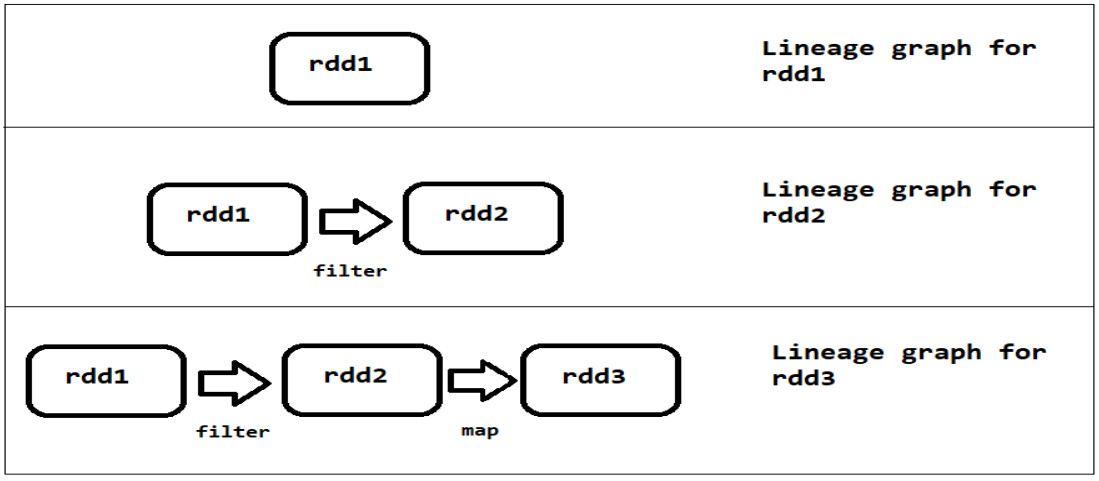
- group By Key does not do a map side combine.
- creates a lot of shuffling which hampers the performance.

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.

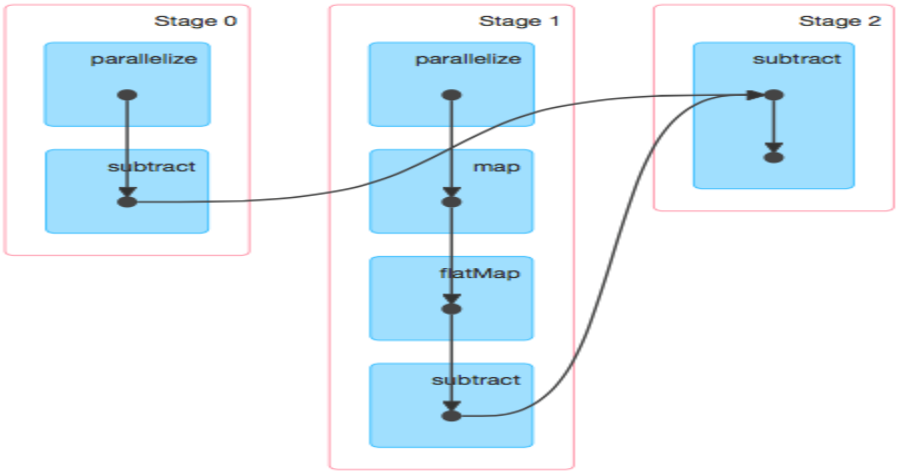
19. What is a Lineage Graph?

- 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.



20. What is the working of DAG in Spark?

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



21. Define Spark 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.
- This lazy evaluation increases the system efficiency.

22. Actions and transformations in spark?

Transformation: is a function that produces new RDD from the existing RDDs.
EX:
Map(),flatMap(),filter()
Actions: An operation takes input ,gives output as a value or collection of values.
Example:
count(),collect(),take(n),top(),countByValue(),reduce(),fold(),aggregate(),foreach()

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 Spark Map vs Flat Map Operation?**

i. Spark Map Transformation

- A map is a transformation operation in Apache Spark. It applies to each element of RDD and it returns the result as new RDD.
- In the Map, operation developer can define his own custom business logic. The same logic will be applied to all the elements of RDD.
- Spark Map function takes one element as input process it according to custom code (specified by the developer) 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 typically have the same number of records.

ii. Spark FlatMap Transformation Operation:

- A flatMap is a transformation operation. It applies to each element of RDD and it returns the result as new RDD.
- It is similar to Map, but FlatMap allows returning 0, 1 or more elements from map function.
- In the FlatMap operation, a developer can define his own custom business logic. The same logic will be applied to all the elements of the RDD.
- A FlatMap function takes one element as input process it according to custom code (specified by the developer) and returns 0 or more element at a time.
- flatMap() transforms an RDD of length N into another RDD of length M.

25. What is Spark mapValues?

- mapValues is only applicable for PairRDDs, meaning RDDs of the form RDD[(A, B)]. In that case, mapValues operates on the value only (the second part of the tuple), while map operates on the entire record (tuple of key and value).

26. Difference between map vs map values?

Mapvalues():

- When you just want to transform the values and keep the keys as-is, it's recommended to use mapValues.
- mapValues, however, preserves any partitioner set on the RDD.

Map ():

- when you just want to transform the both key and use map().
- if you applied any custom partitioning to your RDD (e.g. using partitionBy), using map would "forget" that paritioner (the result will revert to default partitioning) as the keys might have changed;

27. List the types of Deploy Modes in Spark.

There are mainly2 deploy modes in Spark. They are:

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.
- The default deployment mode is client mode.
- 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.

28. Fault tolerant in spark?

The basic semantics of fault tolerance in Apache Spark is, 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.

There are two types of failures – Worker or driver failure. In case if the worker fails, the executors in that worker node will be killed, along with the data in their memory. Using the lineage graph, those tasks will be accomplished in any other worker nodes. The data is also replicated to other worker nodes to achieve fault tolerance. There are two cases:

1. Data received and replicated – Data is received from the source, and replicated across worker nodes. In the case of any failure, the data replication will help achieve fault tolerance.

2. Data received but not yet replicated – Data is received from the source but buffered for replication. In the case of any failure, the data needs to be retrieved from the source.

29. Define Executor Memory in Spark

- Every Spark applications have one allocated executor on each worker node it runs.
- The executor memory is a measure of the memory consumed by the worker node that the application utilizes.
- The applications developed in Spark have the same fixed cores count and fixed heap size defined for spark executors.
- **Heap size**: refers to the memory of the Spark executor that is controlled by making use of the property spark.executor.memory that belongs to the-executor-memory flag.

30. 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.

31. 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)

32. 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 table without hive integration
2. On Single Node Spark cluster from Spark 2 Data stored in Local file system & Metadata will be stored in embedded derby database
3. we can create both spark session and spark context

33. 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.

34. Spark default Ram and Cores

- default ram is - 1 GB.
  default cores are - local/local [1] - 1 core will be added/allocated.
- - local[n] - n cores will be added/allocated
- - local [*] - all available cores in cluster
- from pyspark.sql import sparkSession
- spark=sparkSession.builder.master(local[*]).appName('demo').getOrCreate()

35. 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

36. How to optimize code and performance in spark?

The bottleneck for these spark optimization computations can be CPU, memory, Network bandwidth or any resource in the cluster.

1. **Serialization**
   - Serialization plays an important role in the performance of any distributed application and we know that 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. To set the Kryo serializer as part of a Spark job, we need to set a configuration property, which is org.apache.spark.serializer.KryoSerializer.
2. **API selection**

- Spark introduced three types of API to work upon – RDD, DataFrame, DataSet
- RDD is used for low level operation with less optimization
- DataFrame is best choice in most cases due to its catalyst optimizer and low garbage collection (GC) overhead.
- Dataset is highly type safe and use encoders. It uses Tungsten for serialization in binary format.

3. Advance Variable

- Spark comes with 2 types of advanced variables – Broadcast and Accumulator.
- Broadcasting plays an important role while tuning your spark job.
- Suppose you have a situation where one data set is very small and another data set is quite large, and you want to perform the join operation between these two.
- In that case, we should go for the broadcast join so that the small data set can fit into your broadcast variable.
- The syntax to use the broadcast variable is df1.join(broadcast(df2)). Here we have a second dataframe that is very small and we are keeping this data frame as a broadcast variable.

4. Cache and Persist

- Spark provides its own caching mechanism like Persist() and Caching().
- Persist and Cache mechanisms will store the data set into the memory whenever there is requirement, where you have a small data set and that data set is being used multiple times in your program.
- Cache() – Always in Memory
- Persist() – Memory and disks
- If we apply RDD.Cache() it will always store the data in memory, and if
- we apply RDD.Persist() then some part of data can be stored into the memory some can be stored on the disk.

5. By Key Operation

As we know during our transformation of Spark we have many By Key operations. By Key operations generate lot of shuffle. Shuffles are heavy operation because they consume a lot of memory. While coding in Spark, a user should always try to avoid any shuffle operation because the shuffle operation will degrade the performance. If there is high shuffling then a user can get the

error out of memory. In this case, to avoid that error, a user should increase the level of parallelism.
Instead of groupBy, a user should go for the reduceByKey .

6. **File Format selection**
    - Spark supports many formats, such as CSV, JSON, XML, PARQUET, ORC, AVRO, etc.
    - Spark jobs can be optimized by choosing the parquet file with snappy compression which gives the high performance and best analysis.
    - Parquet file is native to Spark which carries the metadata along with its footer.
    - whenever you create any parquet file, you will see .metadata file on the same directory along with the data file.

7. **Garbage Collection Tuning**
    - As we know underneath our Spark job is running on the JVM platform so JVM garbage collection can be a problem when you have large collection of unused objects.
    - The first step in GC tuning is to collect statistics by choosing – verbose while submitting spark jobs.
    - In an ideal situation we try to keep GC overheads < 10% of heap memory.

8. **Level of Parallelism**
    - In any distributed environment parallelism plays very important role while tuning your Spark job.
    - Whenever a Spark job is submitted, it creates the desk that will contain stages, and the tasks depend upon partition so Every partition or task requires a single core for processing.
    - There are two ways to maintain the parallelism:
    - **Repartition**: Gives equal number of partitions with high shuffling
    - **Coalesce:** Generally reduces the number of partitions with less shuffling.

These factors for spark optimization, if properly used, can

- Eliminate the long-running job process
- Correction execution engine
- Improves performance time by managing resources

37. my spark job running slow in production, how to improve speed?

When a Spark job is running slow in production, there can be several factors contributing to the performance issue. Here are some steps you can take to diagnose and address the problem:

1. **Check resource allocation:** Ensure that your Spark job has been allocated sufficient resources, including CPU cores, memory, and disk space. Inadequate resource allocation can significantly impact performance. Consider increasing the allocated resources if necessary.

2. **Analyze the execution plan:** Examine the execution plan of your Spark job to understand the stages and tasks involved. This can help identify any inefficient or resource-intensive operations, such as unnecessary shuffling, large joins, or data skew.

3. **Optimize data partitioning:** Ensure that the data is properly partitioned to leverage Spark's parallel processing capabilities. If the data is not evenly distributed across partitions, it can lead to data skew and performance degradation. Consider repartitioning or using appropriate partitioning techniques based on your data characteristics and processing requirements.

4. **Tune Spark configurations:** Adjusting various Spark configuration settings can improve performance. For example, you can increase the memory allocated to Spark executors (`spark.executor.memory`), adjust the number of executor cores (`spark.executor.cores`), or modify the shuffle memory fraction (`spark.shuffle.memoryFraction`) to optimize memory usage.

5. **Use appropriate caching:** If you have repetitive operations or multiple stages depending on the same intermediate data, consider caching the data in memory (`DataFrame.cache()` or `RDD.cache()`) to avoid recomputation. This can save processing time, especially for iterative algorithms or complex workflows.

6. **Enable data compression:** If the size of the data being processed is substantial, enabling compression techniques such as Snappy or Gzip can reduce disk I/O and network transfer overhead, thereby improving performance.

7. **Consider data partition pruning:** If your job involves filtering or aggregating data based on specific criteria, leverage partition pruning techniques to minimize the amount of data to be processed. This can be achieved by partitioning your data based on relevant attributes and utilizing predicates while querying the data.

8**. Profile and optimize code:** Profile your Spark code to identify any performance bottlenecks. Use tools like Spark's built-in monitoring interfaces or external profilers to analyze the execution time and resource usage of different operations. Once you identify the problematic areas, optimize the code, apply appropriate algorithms, or consider alternative approaches to improve performance.

9. **Parallelize and distribute work:** If your Spark job involves iterative or expensive computations, explore opportunities to parallelize and distribute the work. Utilize distributed algorithms or libraries that support parallel processing, such as Spark MLlib or GraphX.

10**. Upgrade Spark version:** Consider upgrading to the latest stable version of Apache Spark. Newer versions often include performance optimizations and bug fixes that can enhance the overall efficiency and speed of your Spark jobs.

38. 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

39. 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)

- Number of available executors = 150/5 = 30

NUMBER OF EXECUTORS PER ONE NODE = (**NUMBER OF AVAILABLE EXECUTORS/ NO.OF NODES IN CLUSTER)**

- Number of executors per one node = 30/10 = 3

MEMORY PER ONE EXECUTOR = (AVAILABLE MEMORY IN ONE NODE / NUMBER OF EXECUTORS per ONE NODE)

- Memory per executor = 64GB/3 = 21GB
- Counting off heap overhead = 7% of 21GB = 3GB.
- So, **ACTUAL EXECUTOR-MEMORY** = 21 - 3 = 18GB
- Leaving 1 executor for ApplicationManager .
- So then **NO.OF EXECUTORS** = 29

So, recommended config is: 29 executors, 18GB memory each and 5 cores each!!

40. How to write an udf in spark?

Spark SQL UDF (User Defined Function) is the most useful feature of Spark SQL & DataFrame which extends the Spark build in capabilities.
UDF's are the most expensive operations hence use them only you have no choice and when essential

For example if you wanted to convert the every first letter of a word in a sentence to capital case, spark build-in features does't have this function hence you can create it as UDF and reuse this as needed on many Data Frames. UDF's are once created they can be re-use on several DataFrame's and SQL expressions.

41. **What are the important components of the Spark ecosystem?**

Apache Spark has 3 main categories that comprise its ecosystem. Those are:

- **Language support:** Spark can integrate with different languages to applications and perform analytics. These languages are Java, Python, Scala, and R.
- **Core Components:** Spark supports 5 main core components. There are Spark Core, Spark SQL, Spark Streaming, Spark MLlib, and GraphX.
- **Cluster Management:** Spark can be run in 3 environments. Those are the Standalone cluster, Apache Mesos, and YARN.

42. **What are the features of Apache Spark?**

- **High Processing Speed:** Apache Spark helps in the achievement of a very high processing speed of data by reducing read-write operations to disk. The speed is almost 100x faster while performing in-memory computation and 10x faster while performing disk computation.
- **Dynamic Nature:** Spark provides 80 high-level operators which help in the easy development of parallel applications.
- **In-Memory Computation:** The in-memory computation feature of Spark due to its DAG execution engine increases the speed of data processing. This also supports data caching and reduces the time required to fetch data from the disk.
- **Reusability:** Spark codes can be reused for batch-processing, data streaming, running ad-hoc queries, etc.
- **Fault Tolerance:** Spark supports fault tolerance using RDD. Spark RDDs are the abstractions designed to handle failures of worker nodes which ensures zero data loss.
- **Stream Processing:** Spark supports stream processing in real-time. The problem in the earlier MapReduce framework was that it could process only already existing data.
- **Lazy Evaluation:** When Spark operates on any dataset, it remembers the instructions. When a transformation such as a map() is called on an RDD, the operation is not performed instantly. Transformations in Spark are not evaluated until you perform an action, which aids in optimizing the overall data processing workflow, known as lazy evaluation.This lazy evaluation increases the system efficiency.
- **Hadoop Integration:** Spark also supports the Hadoop YARN cluster manager thereby making it flexible.
- Supports Spark GraphX for graph parallel execution, Spark SQL, libraries for Machine learning, etc.
- **Cost Efficiency:** Apache Spark is considered a better cost-efficient solution when compared to Hadoop as Hadoop required large storage and data centers while data processing and replication.
- **Active Developer's Community:** Apache Spark has a large developers base involved in continuous development. It is considered to be the most important project undertaken by the Apache community.

43. Explain how Spark runs applications with the help of its architecture.

Spark applications run as independent processes that are coordinated by the SparkSession object in the driver program. The resource manager or cluster manager assigns tasks to the worker nodes with one task per partition. Iterative algorithms apply operations repeatedly to the data so they can benefit from caching datasets across iterations. A task applies its unit of work to the dataset in its partition and

outputs a new partition dataset. Finally, the results are sent back to the driver application or can be saved to the disk.

44. What are the different cluster managers available in Apache Spark?

- **Standalone Mode:** By default, applications submitted to the standalone mode cluster will run in FIFO order, and each application will try to use all available nodes. You can launch a standalone cluster either manually, by starting a master and workers by hand, or use our provided launch scripts. It is also possible to run these daemons on a single machine for testing.
- **Apache Mesos:** Apache Mesos is an open-source project to manage computer clusters, and can also run Hadoop applications. The advantages of deploying Spark with Mesos include dynamic partitioning between Spark and other frameworks as well as scalable partitioning between multiple instances of Spark.
- **Hadoop YARN:** Apache YARN is the cluster resource manager of Hadoop
- **Kubernetes:** Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

45. Under what scenarios do you use Client and Cluster modes for deployment?

- In case the client machines are not close to the cluster, then the Cluster mode should be used for deployment. This is done to avoid the network latency caused while communication between the executors which would occur in the Client mode. Also, in Client mode, the entire process is lost if the machine goes offline.
- If we have the client machine inside the cluster, then the Client mode can be used for deployment. Since the machine is inside the cluster, there won't be issues of network latency and since the maintenance of the cluster is already handled, there is no cause of worry in cases of failure.

46. What are receivers in Apache Spark Streaming?

Receivers are those entities that consume data from different data sources and then move them to Spark for processing.

Each receiver is configured to use up only a single core. The receivers are made to run on various executors to accomplish the task of data streaming. There are two types of receivers depending on how the data is sent to Spark:

Reliable receivers: Here, the receiver sends an acknowledgement to the data sources post successful reception of data and its replication on the Spark storage space.

Unreliable receiver: Here, there is no acknowledgement sent to the data sources..

47. What are the data formats supported by Spark?

Spark supports both the raw files and the structured file formats for efficient reading and processing. File formats like paraquet, JSON, XML, CSV, RC, Avro, TSV, etc are supported by Spark.

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

- The process of redistribution of data across different partitions which might or might not cause data movement across the JVM processes or the executors on the separate machines is known as shuffling/repartitioning.
- Partition is nothing but a smaller logical division of data.
- It is to be noted that Spark has no control over what partition the data gets distributed across.
- The shuffle operation is implemented differently in Spark compared to Hadoop.
- Shuffling has 2 important compression parameters:
  spark.shuffle.compress – checks whether the engine would compress shuffle outputs or not
  spark.shuffle.spill.compress – decides whether to compress intermediate shuffle spill files or not
- It occurs while joining two tables or while performing byKey operations such as GroupByKey or ReduceByKey

49. What is YARN in Spark?

- YARN is a generic resource-management framework for distributed workloads; in other words, a cluster-level operating system.
- Scheduling on a YARN cluster; it is either a single job or a DAG of jobs (jobs here could mean a Spark job, an Hive query or any similar constructs).

50. What is Spark Streaming and how is it implemented in Spark?

- Spark Streaming is one of the most important features provided by Spark.
- It is nothing but a Spark API extension for supporting stream processing of data from different sources.
- Data from sources like Kafka,Flume, etc are processed and pushed to various destinations like databases, dashboards, machine learning APIs, or as simple as file systems. The data is divided into various streams (similar to batches) and is processed accordingly.

- Spark streaming supports highly scalable, fault-tolerant continuous stream processing which is mostly used in cases like fraud detection, website monitoring, website click baits, IoT (Internet of Things) sensors, etc.

- Spark Streaming first divides the data from the data stream into batches of X seconds which are called Dstreams or Discretized Streams. They are internally nothing but a sequence of multiple RDDs.

- The Spark application does the task of processing these RDDs using various Spark APIs and the results of this processing are again returned as batches. The following diagram explains the workflow of the spark streaming process.

51. What are the functions of SparkCore?

SparkCore is the main engine that is meant for large-scale distributed and parallel data processing. The Spark core consists of the distributed execution engine that offers various APIs in Java, Python, and Scala for developing distributed ETL applications.

The Spark Core mainly deals with the Rdd and Spark Context.

52. What are the various functionalities supported by Spark Core?

Spark Core is the engine for parallel and distributed processing of large data sets.

The various functionalities supported by Spark Core include:

- Scheduling and monitoring jobs
- Memory management
- Fault recovery
- Task dispatching

53. **What is Spark Context?**

Spark Context: Represents the 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.

54. What do you understand by worker node?

Worker nodes are those nodes that run the Spark application in a cluster. The Spark driver program listens for the incoming connections and accepts them from the executors addresses them to the worker nodes for execution. A worker node is like a slave node where it gets the work from its master node and actually executes them. The worker nodes do data processing and report the resources used to the master. The master decides what amount of resources needs to be allocated and then based on their availability, the tasks are scheduled for the worker nodes by the master.

55. What are some of the demerits of using Spark in applications?

Despite Spark being the powerful data processing engine, there are certain demerits to using Apache Spark in applications. Some of them are:

Spark makes use of more storage space when compared to MapReduce or Hadoop which may lead to certain memory-based problems.

Care must be taken by the developers while running the applications. The work should be distributed across multiple clusters instead of running everything on a single node.

Since Spark makes use of "in-memory" computations, they can be a bottleneck to cost-efficient big data processing.

While using files present on the path of the local filesystem, the files must be accessible at the same location on all the worker nodes when working on cluster mode as the task execution shuffles between various worker nodes based on the resource availabilities. The files need to be copied on all worker nodes or a separate network-mounted file-sharing system needs to be in place. One of the biggest problems while using Spark is when using a large number of small files. When Spark is used with Hadoop, we know that HDFS gives a limited number of large files instead of a large number of small files. When there is a large number of small gzipped files, Spark needs to uncompress these files by keeping them on its memory and network. So large amount of time is spent in burning core capacities for unzipping the files in sequence and performing partitions of the resulting RDDs to get data in a manageable format which would require extensive shuffling overall. This impacts the performance of Spark as much time is spent preparing the data instead of processing them. Spark doesn't work well in multi-user environments as it is not capable of handling many users concurrently.

56. How can the 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. There are various ways in which these can be minimized. They are:

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.

57. What is SchemaRDD in Spark RDD?

SchemaRDD is an RDD consisting of row objects that are wrappers around integer arrays or strings that has schema information regarding the data type of each column. They were designed to ease the lives of developers while debugging the

code and while running unit test cases on the SparkSQL modules. They represent the description of the RDD which is similar to the schema of relational databases. SchemaRDD also provides the basic functionalities of the common RDDs along with some relational query interfaces of SparkSQL.

Consider an example. If you have an RDD named Person that represents a person's data. Then SchemaRDD represents what data each row of Person RDD represents. If the Person has attributes like name and age, then they are represented in SchemaRDD.

58. What module is used for implementing SQL in Apache Spark?

Spark provides a powerful module called SparkSQL which performs relational data processing combined with the power of the functional programming feature of Spark. This module also supports either by means of SQL or Hive Query Language. It also provides support for different data sources and helps developers write powerful SQL queries using code transformations.

The four major libraries of SparkSQL are:

Data Source API

DataFrame API

Interpreter & Catalyst Optimizer

SQL Services

Spark SQL supports the usage of structured and semi-structured data in the following ways:

Spark supports DataFrame abstraction in various languages like Python, Scala, and Java along with providing good optimization techniques.

SparkSQL supports data read and writes operations in various structured formats like JSON, Hive, Parquet, etc.

SparkSQL allows data querying inside the Spark program and via external tools that do the JDBC/ODBC connections.

It is recommended to use SparkSQL inside the Spark applications as it empowers the developers to load the data, query the data from databases and write the results to the destination.

59. What are the different persistence levels in Apache Spark?

Spark persists intermediary data from different shuffle operations automatically. But it is recommended to call the persist() method on the RDD. There are different persistence levels for storing the RDDs on memory or disk or both with different levels of replication. The persistence levels available in Spark are:

MEMORY_ONLY: This is the default persistence level and is used for storing the RDDs as the deserialized version of Java objects on the JVM. In case the RDDs are huge and

do not fit in the memory, then the partitions are not cached and they will be recomputed as and when needed.

MEMORY_AND_DISK: The RDDs are stored again as deserialized Java objects on JVM. In case the memory is insufficient, then partitions not fitting on the memory will be stored on disk and the data will be read from the disk as and when needed.
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.
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.
The syntax for using persistence levels in the persist() method is:
df.persist(StorageLevel.<level_value>)
Persistence Level Space Consumed CPU time InMEMORY_ONLY High Low YMEMORY_ONLY_SER Low High YMEMORY_AND_DISK High Medium SMEMORY_AND_DISK_SER Low High SDISK_ONLY Low High NOFF_HEAP Low HighYh

60. What are Sparse Vectors? How are they different from dense vectors?

Sparse vectors consist of two parallel arrays where one array is for storing indices and the other for storing values. These vectors are used to store non-zero values for saving space.
val sparseVec: Vector = Vectors.sparse(5, Array(0, 4), Array(1.0, 2.0))
In the above example, we have the vector of size 5, but the non-zero values are there only at indices 0 and 4.
Sparse vectors are particularly useful when there are very few non-zero values. If there are cases that have only a few zero values, then it is recommended to use dense vectors as usage of sparse vectors would introduce the overhead of indices which could impact the performance.
Dense vectors can be defines as follows:
val denseVec
=Vectors.dense(4405d,260100d,400d,5.0,4.0,198.0,9070d,1.0,1.0,2.0,0.0)
Usage of sparse or dense vectors does not impact the results of calculations but when used inappropriately, they impact the memory consumed and the speed of calculation.

61. How are automatic clean-ups triggered in Spark for handling the accumulated metadata?

The clean-up tasks can be triggered automatically either by setting spark.cleaner.ttl parameter or by doing the batch-wise division of the long running jobs and then writing the intermediary results on the disk.

62. Explain Caching in Spark Streaming.

Caching also known as Persistence is an optimization technique for Spark computations. Similar to RDDs, DStreams also allow developers to persist the stream's data in memory. That is, using the persist() method on a DStream will automatically persist every RDD of that DStream in memory. It helps to save interim partial results so they can be reused in subsequent stages.

The default persistence level is set to replicate the data to two nodes for fault-tolerance, and for input streams that receive data over the network such as Kafka, Flume.

Caching using cache method:

val cacheDf = dframe.cache()

Caching using persist method:

val persistDf = dframe.persist(StorageLevel.MEMORY_ONLY)

The main advantages of caching are:

Cost efficiency: Since Spark computations are expensive, caching helps to achieve reusing of data and this leads to reuse computations which can save the cost of operations.

Time-efficient: The computation reusage leads to saving a lot of time. More Jobs Achieved: By saving time of computation execution, the worker nodes can perform/execute more jobs.

63. Define Piping in Spark.

Pipe operator in Spark, **allows developer to process RDD data using external applications**. Sometimes in data analysis, we need to use an external library which may not be written using Java/Scala. ... In that case, spark's pipe operator allows us to send the RDD data to the external application.

64. What API is used for Graph Implementation in Spark?

Spark provides a powerful API called GraphX that extends Spark RDD for supporting graphs and graph-based computations. The extended property of Spark RDD is called as Resilient Distributed Property Graph which is a directed multi-graph that has multiple parallel edges. Each edge and the vertex has associated user-defined properties. The presence of parallel edges indicates multiple relationships between the same set of vertices. GraphX has a set of operators such as subgraph, mapReduceTriplets, joinVertices, etc that can support graph computation. It also includes a large collection of graph builders and algorithms for simplifying tasks related to graph analytics.

65. How can you achieve machine learning in Spark?

Spark provides a very robust, scalable machine learning-based library called MLlib. This library aims at implementing easy and scalable common ML-based algorithms and has the features like classification, clustering, dimensional reduction, regression filtering, etc.

66. What makes Spark good at low latency workloads like graph processing and Machine Learning?

Apache Spark stores data in-memory for faster processing and building machine learning models. Machine Learning algorithms require multiple iterations and different conceptual steps to create an optimal model. Graph algorithms traverse through all the nodes and edges to generate a graph. These low latency workloads that need multiple iterations can lead to increased performance.

67. How can you connect Spark to Apache Mesos?

There are a total of 4 steps that can help you connect Spark to Apache Mesos.

- Configure the Spark Driver program to connect with Apache Mesos
- Put the Spark binary package in a location accessible by Mesos
- Install Spark in the same location as that of the Apache Mesos
- Configure the spark.mesos.executor.home property for pointing to the location where Spark is installed

68. How do you convert a Spark RDD into a DataFrame?

There are 2 ways to convert a Spark RDD into a DataFrame:
Using the helper function – toDF
import com.mapr.db.spark.sql._
val df = sc.loadFromMapRDB(<table-name>)
.where(field(“first_name”) === “Peter”)
.select(“_id”, “first_name”).toDF()

Using SparkSession.createDataFrame You can convert an RDD[Row] to a DataFrame by calling createDataFrame on a SparkSession object def createDataFrame(RDD, schema:StructType)

69. How to register a temporary table in spark sql?

Ans:- When we are creating the data frame by loading the data into it using SQLContext object. This is treated as a temporary table. Because the scope of the data frame is to particular session.

70. Will use HiveContext object rather than SparkContext Object?

Ans:-In addition to the basic SQLContext, you can also create a HiveContext, which provides a superset of the functionality provided by the basic SQLContext. Additional features include the ability to write queries using the more complete HiveQL parser, access to Hive UDFs, and the ability to read data from Hive tables. To use a HiveContext, you do not need to have an existing Hive setup, and all of the data sources available to a SQLContext are still available. HiveContext is only packaged separately to avoid including all of Hive's dependencies in the default Spark build. If these dependencies are not a problem for your application then using HiveContext is recommended for the 1.3 release of Spark.

The specific variant of SQL that is used to parse queries can also be selected using the spark.sql.dialect option. This parameter can be changed using either the setConf method on a SQLContext or by using a SET key=value command in SQL. For a SQLContext, the only dialect available is "sql" which uses a simple SQL parser provided by Spark SQL. In a HiveContext, the default is "hiveql", though "sql" is also available. Since the HiveQL parser is much more complete, this is recommended for most use cases.

71. **If an Rdd is having the 10 gb file .. I performed some operation on that and removed the las few lines of the textfile ? so it save into another RDD is it correct to maintain the both the RDD's how you handle it?**

    Ans:- Where RDD is an lazy-evaluation. Since when you call the action it performs the operations like creating and loading the data into RDD.

    When you want to make that RDD as permanent you need to call the persist method. To save the RDD as permanent and use saveAsTextFile to save that RDD data into Hard disk..

72. **By the above scenario how we can say the project memory capacity** ?

    Ans:- The Project –size depends upon the number of transformations and actions. So in real-time we all the people using the cloud based memory so in that case it will automatically allocates the more or less memory as it needed to perform that transformation.