Spark Heavy Questions:

1. What is the business case of your pipeline (the problem your company is trying to solve)?

2. What is your architecture of your pipeline(s)?

3. What technology stack have you used, and why?

4. What is your specific contribution within the pipeline?

5. Difference between RDD, data frame, and data set?

- 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.
- 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.
- 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.

6. What is the catalyst optimizer?

Catalyst optimizer optimizes logical plan of the SQL query which run with Spark Sql. Its a set of rules which apply on the sqlquery to rewrite it in better way to gain performance. For example, sql WHERE filters will be applied first on initial data if possible instead applying at the end.

7. What is the project tungsten and when do you use it?

The goal of tungsten substantially improve the memory and CPU efficiency of the Spark applications and push the limits of the underlying hardware. 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. Project Tungsten aims at substantially reducing the usage of JVM objects (and therefore JVM garbage collection) by introducing its own off-heap binary memory management.

8. What's the size of the data?

9. How do you calculate the size of your data?

A back-of-the-envelope calculation for the size of a dataset is

$$\text{number of megabytes} = M = \frac{N*V*W + 4*N}{1024^2}$$

where

N = number of observations
V = number of variables
W = average width in bytes of a variable

10. How many nodes are in the cluster?

By default, 3-4 nodes should be enough in a cluster. However, with every version of spark there are minor improvements in performance. It also depends on which version of Scala/Python you have compiled spark with and which version of JVM you're using. These factors, however, do not effect the speed of your program so much as the program you write and data-time demand.

11. How do you calculate the number of executors and memory inside your cluster?

- Hadoop/Yarn/OS Deamons: When we run spark application using a cluster manager like Yarn, there'll be several daemons that'll run in the background like NameNode, Secondary NameNode, DataNode, JobTracker and TaskTracker. So, while specifying num-executors, we need to make sure that we leave aside enough cores (~1 core per node) for these daemons to run smoothly.
- Yarn ApplicationMaster (AM): ApplicationMaster is responsible for negotiating resources from the ResourceManager and working with the NodeManagers to execute and monitor the containers and their resource consumption. If we are running spark on yarn, then we need to budget in the resources that AM would need (~1024MB and 1 Executor).
- HDFS Throughput: HDFS client has trouble with tons of concurrent threads. It was observed that HDFS achieves full write throughput with ~5 tasks per executor. So it's good to keep the number of cores per executor below that number.
- MemoryOverhead: Following picture depicts spark-yarn-memory-usage

---

Full memory requested to yarn per executor =

spark-executor-memory + spark.yarn.executor.memoryOverhead.

spark.yarn.executor.memoryOverhead =

Max(384MB, 7% of spark.executor-memory)

So, if we request 20GB per executor, AM will actually get 20GB + memoryOverhead = 20 + 7% of 20GB = ~23GB memory for us.

- Running executors with too much memory often results in excessive garbage collection delays.

- Running tiny executors (with a single core and just enough memory needed to run a single task, for example) throws away the benefits that come from running multiple tasks in a single JVM.

12. What is data skewness?

Data skew means that data distribution is uneven or asymmetric. Symmetry means that one half of the distribution is a mirror image of the other half.

13. How do you solve the problem of skewness?

Broadcast Join solution is not scalable. It has a hardcoded limitation on size of broadcasted tables. Sort Merge Join with Keys Salting can be scaled to any size. Also, Broadcast join is easier to maintain since no additional code is needed. Just add a 'broadcast' hint to the join operation of Apache Spark and you are good to go.

14. What is a broadcast variable?

Broadcast variables are read-only variables that are distributed across worker nodes in-memory instead of shipping a copy of data with tasks. Broadcast variables are mostly used when the tasks across multiple stages require the same data or when caching the data in the deserialized form is required.

15. What kind of transformations have you been doing in your data?

Two most basic type of transformations is a map(), filter(). After the transformation, the resultant RDD is always different from its parent RDD. It can be smaller (e.g. filter, count, distinct, sample), bigger (e.g. flatMap(), union(), Cartesian()) or the same size (e.g. map). There are two types of transformations:

- Narrow transformation – In Narrow transformation, all the elements that are required to compute the records in single partition live in the single partition of parent RDD. A limited subset of partition is used to calculate the result. Narrow transformations are the result of map(), filter().
- Wide transformation – In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. Wide transformations are the result of groupbyKey() and reducebyKey().

16. Difference between groupByKey versus reduceByKey?

- When a groupByKey is called on a RDD pair the data in the partitions are shuffled over the network to form a key and list of values.
- The reduceByKey works much better on a large dataset as compared to. That's because Spark knows it can combine output with a common key on each partition before shuffling the data.

17. What is shuffling?

Shuffling is a process of redistributing data across partitions (aka repartitioning) that may or may not cause moving data across JVM processes or even over the wire (between executors on separate machines). Shuffling is the process of data transfer between stages.

18. How do you decrease shuffling?

- Just set SPARK_LOCAL_DIRS on standalone and mesos based schedulers and LOCAL_DIRS on YARN based schedulers to point to a different location than the default.
- Make sure that location is an SSD and of a large enough size to fit your workload.
- Determine what your workloads LOOK like before you deploy them. Shuffle operations aren't free and this is still distributed computing, if these operations are performance sensitive you should think long and hard about how you have your data partitioned from the get go.

19. Common issues with memory in Spark?

If we were to got all Spark developers to vote, out of memory (OOM) conditions would surely be the number one problem everyone has faced. This comes as no big surprise as Spark's architecture is memory-centric. Some of the most common causes of OOM are:

- Incorrect usage of Spark
- High concurrency
- Inefficient queries
- Incorrect configuration

20. Difference between coalesce and repartition (or partition)?

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.

21. What is the standard garbage collector for Spark?

Because Spark can store large amounts of data in memory, it has a major reliance on Java's memory management and garbage collection (GC). New initiatives like Project Tungsten will simplify and optimize memory management in future Spark versions.

22. How to read a file in Spark?

textFile() method is used to read a text file from HDFS, S3 and any Hadoop supported file system, this method takes the path as an argument and optionally takes a number of partitions as the second argument.

23. Difference between Spark and MapReduce?

The key difference between MapReduce and Apache Spark is explained below:

- MapReduce is strictly disk-based while Apache Spark uses memory and can use a disk for processing.
- MapReduce and Apache Spark both have similar compatibility in terms of data types and data sources.
- The primary difference between MapReduce and Spark is that MapReduce uses persistent storage and Spark uses Resilient Distributed Datasets.

- Hadoop MapReduce is meant for data that does not fit in the memory whereas Apache Spark has a better performance for the data that fits in the memory, particularly on dedicated clusters.
- Hadoop MapReduce can be an economical option because of Hadoop as a service and Apache Spark is more cost effective because of high availability memory
- Apache Spark and Hadoop MapReduce both are failure tolerant but comparatively Hadoop MapReduce is more failure tolerant than Spark.
- Hadoop MapReduce requires core java programming skills while Programming in Apache Spark
- Spark is able to execute batch-processing jobs between 10 to 100 times faster than the MapReduce Although both the tools are used for processing Big Data.

24. What is lazy evaluation?

Lazy evaluation in Spark means that the execution will not start until an action is triggered. In Spark, the picture of lazy evaluation comes when Spark transformations occur. Transformations are lazy in nature meaning when we call some operation in RDD, it does not execute immediately.

25. What is a trait?

Traits can be mixed into objects to add commonly used methods or values. We can define a SparkSessionWrapper trait that defines a spark variable to give objects easy access to the SparkSession object.

26. What are case classes and what are they used for?

The case class defines the schema of the table. The names of the arguments to the case class are read using reflection and they become the names of the columns.

27. What is the DAG?

(Directed Acyclic Graph) DAG in Apache Spark is a set of Vertices and Edges, where vertices represent the RDDs and the edges represent the Operation to be applied on RDD. In Spark DAG, every edge directs from earlier to later in the sequence. On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task.

28. Define the lineage of a Spark job.

RDD Lineage (aka RDD operator graph or RDD dependency graph) actually is a graph of all the parent RDDs of an RDD. It is built as a consequence of applying transformations to the RDD and creates a logical execution plan.

29. How do you submit your job to a Spark cluster?

To submit Spark jobs to an EMR cluster from a remote machine, the following must be true:

- Network traffic is allowed from the remote machine to all cluster nodes.
- All Spark and Hadoop binaries are installed on the remote machine.
- The configuration files on the remote machine point to the EMR cluster.

30. How to create RDDs in Spark?

There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

31. How to convert RDD to Dataframes?

Converting Spark RDD to DataFrame can be done using toDF(), createDataFrame() and transforming rdd[Row] to the data frame.

32. What are the partitions in Spark

A partition in spark is an atomic chunk of data (logical division of data) stored on a node in the cluster. Partitions are basic units of parallelism in Apache Spark. RDDs in Apache Spark are collection of partitions.

33. What is an action?

Actions are Spark RDD operations that give non-RDD values. The values of action are stored to drivers or to the external storage system. It brings laziness of RDD into motion. An action is one of the ways of sending data from Executer to the driver.

34. What are transformations?

Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. Thus, the so input RDDs, cannot be changed since RDD are immutable in nature.

35. What is a pair RDD?

pairRDD operations (such as map, reduceByKey etc) produce key,value pairs. Whereas operations on RDD(such as flatMap or reduce) gives you a collection of values or a single value. pairRDD operations are also applied on each key/element in parallel.Operations on RDD (like flatMap) are applied to the whole collection

36. What is the Driver?

The driver is the process where the main method runs. First it converts the user program into tasks and after that it schedules the tasks on the executors.

37. What is an executor?

Executors are worker nodes' processes in charge of running individual tasks in a given Spark job. They are launched at the beginning of a Spark application and typically run for the entire lifetime of an application. Once they have run the task they send the results to the driver. They also provide in-memory storage for RDDs that are cached by user programs through Block Manager.

38. Tell me what are the submissions modes for a Spark job?

39. Type of cluster managers in Spark?

The system currently supports several cluster managers:

- Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.

- Apache Mesos – a general cluster manager that can also run Hadoop MapReduce and service applications.
- Hadoop YARN – the resource manager in Hadoop 2.
- Kubernetes – an open-source system for automating deployment, scaling, and management of containerized applications.

40. What is a sparse vector?

A sparse vector is used for storing non-zero entries for saving space. It has two parallel arrays: (1) One for indices and (2) The other for values

41. What is an accumulator?

Accumulators are variables that are only "added" to through an associative 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.

42. What are sliding windows?

As window slides over a source DStream, the source RDDs that fall within the window are combined. It also operated upon which produces spark RDDs of the windowed DStream. Basically, any Spark window operation requires specifying two parameters:

- Window length – It defines the duration of the window
- Sliding interval – It defines the interval at which the window operation is performed

43. Define caching in Spark.

Caching RDDs in Spark: It is one mechanism to speed up applications that access the same RDD multiple times. When to use caching: As suggested in this post, it is recommended to use caching in the following situations:

- RDD re-use in iterative machine learning applications
- RDD re-use in standalone Spark applications
- When RDD computation is expensive, caching can help in reducing the cost of recovery in the case one executor fails

44. Levels of persistence in Spark?

1. MEMORY_ONLY: In this level, RDD object is stored as a de-serialized Java object in JVM. If an RDD doesn't fit in the memory, it will be recomputed.
2. MEMORY_AND_DISK: In this level, RDD object is stored as a de-serialized Java object in JVM. If an RDD doesn't fit in the memory, it will be stored on the Disk.
3. MEMORY_ONLY_SER: In this level, RDD object is stored as a serialized Java object in JVM. It is more efficient than de- serialized object.

4. MEMORY_AND_DISK_SER: In this level, RDD object is stored as a serialized Java object in JVM. If an RDD doesn't fit in the memory, it will be stored on the Disk.

5. DISK_ONLY: In this level, RDD object is stored only on Disk.

45. Difference between persistence and cache?

RDD cache() method default saves it to memory (MEMORY_ONLY) whereas persist() method is used to store it to user-defined storage level.

- MEMORY_ONLY - In this storage level, RDD is stored as deserialized Java object in the JVM. If the size of RDD is greater than memory, then it will not cache some partition and recompute them next time whenever needed. In this level the space used for storage is very high, the CPU computation time is low, the data is stored in-memory. It does not make use of the disk.

46. What are check points?

Checkpointing saves an RDD to a reliable storage system (e.g. HDFS, S3) while forgetting the RDD's lineage completely. As the driver restarts the recovery takes place. Also, there are two types of Apache Spark checkpointing:

- Reliable Checkpointing – It refers to that checkpointing in which the actual RDD is saved in reliable distributed file system, e.g. HDFS. To set the checkpoint directory call: SparkContext.setCheckpointDir(directory: String). When running on the cluster the directory must be an HDFS path since the driver tries to recover the checkpointed RDD from a local file. While the checkpoint files are actually on the executor's machines.

- Local Checkpointing: In this checkpointing, in Spark Streaming or GraphX we truncate the RDD lineage graph in Spark. In this, the RDD is persisted to local storage in the executor

47. What is fault tolerance in Spark?

In Apache Spark, the data storage model is based on RDD. These RDDs help to achieve fault tolerance through the lineage, which lets RDDs always have information on how to build from other datasets and if any partition of an RDD is lost due to failure, lineage helps build only that particular lost partition.

48. Different types of compression for Spark?

Apache Spark provides a very flexible compression codecs interface with default implementations like GZip, Snappy, LZ4, and ZSTD

49. What is a map, versus a flat map, versus a fold?

- 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.

- 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.

- Fold is an action because it is a wide operation (i.e. shuffle data across multiple partitions and output a single value). It takes function as an input which has two parameters of the same type and outputs a single value of the input type. It is similar to reduce but has one more argument 'ZERO VALUE' (say initial value) which will be used in the initial call on each partition.

50. What is the difference between an input block versus a partition in an RDD?

- When data is read from Data Bricks File System, it is divided into input blocks, which are then sent to different executors. This configuration controls the size of these input blocks when increasing the number of tasks per stage.
- In apache spark, we store data in the form of RDDs which are Resilient Distributed Datasets. They are a collection of various data items that are so huge in size. That big size data cannot fit into a single node. Thus, we need to divide it into partitions across various nodes, spark automatically partitions RDDs. Also, automatically distributes the partitions among different nodes. In spark, the partition is an atomic chunk of data. Simply putting, it is a logical division of data stored on a node over the cluster. In apache spark, partitions are basic units of parallelism and RDDs, in spark are the collection of partitions.