Top 50+ Apache Spark Interview Questions and Answers

1. Best Spark Interview Questions and Answers

The **Big Data** technology is an umbrella term. It is emerging with time. **Apache Hadoop**, **Apache Spark** is the framework for dealing with this. The revenue of Big Data is increasing exponentially. To become a part of Bigdata industry I hope these Top 50+ Apache Spark Interview Questions and Answers will help to get an edge in Bigdata Market.

As a Bigdata professional, it is necessary to know the right buzzword, and the right answer to frequently asked Spark Interview questions and answers. The Apache Spark Interview Questions and answers mentioned below revolves around the concept of **Spark Core**, **Spark Streaming**, **Spark SQL**, **GraphX**, and **MLlib**.
Hope this blog will act as a gateway to your Spark Job.
So, let's explore top Spark Interview Questions and Answers.

2. Top Apache Spark Interview Questions and Answers

Here we are going to discuss the list of Spark interview questions and answers. We have classified these Apache Spark Interview Questions according to Spark ecosystem components-

a. Apache Spark Basic Interview Questions and Answers

Here are some Frequently asked Spark Interview Questions and Answers for freshers and experienced.

Q.1 What is Apache Spark?

Apache Spark is open source, wide range data processing engine. It is data processing engine with high APIs. It allows data worker to execute *streaming, machine learning or SQL* workloads. These jobs need fast iterative access to datasets. Spark provides API in various languages like **Python, R, Scala, Java**. We can run Spark by itself or on various existing cluster manager. There is various deployment option in Spark. For example, **Standalone Deploy Mode, Apache Mesos, Hadoop YARN.**
The design of the Apache is so dynamic that it can integrate with all the **Big Data** tools. For example, Spark can access data from any of the **Hadoop** data source. It can also run in Hadoop data cluster. Spark does not have its own storage system. It relies on **HDFS** or other file storage for storing the data.

Q.2 Why does the picture of Spark come into existence?
To overcome the drawbacks of **Apache Hadoop**, Spark came into the picture. Some of the drawbacks of Hadoop that Apache Spark overcomes are:

- Hadoop used only Java to build applications. Because it uses Java there were some security concerns as Java is prone to cyber crime.
- Apache Hadoop was apt only for batch processing. So, it does not support stream processing which was overcome in Spark.
- Hadoop used disk-based processing which results in slower retrieving of data. Spark overcomes this by in-memory computation.

Q.3 What are the features of Spark?
Some of the features of Apache Spark are:

- The processing speed of Apache Spark is very high.
- Apache Spark is dynamic in nature. There are about 80 high-level operators, thus, using which we can build a parallel application.
- We can reuse code for join stream against historical data or for batch processing.
- Through **RDD** we achieve fault tolerance. Thus, the data recovery is possible in RDD.

- Spark support many languages like **Java, Scala, Python, and R**. Thus, makes it more user-friendly and is dynamic in nature.
- It can run independently and also on other cluster managers like Hadoop YARN.
- Apache Spark is cost effective solution for Big data problem. While Hadoop needs large storage and the large data center during replication.

Q.4 What are the limitations of Spark?

- Does not have its file management system. Thus, it needs to integrate with Hadoop or other cloud-based data platforms.
- In-memory capability can become a bottleneck. Especially when it comes to cost-efficient processing of Bigdata.
- Memory consumption is very high. And the issues for the same are not handled in a user-friendly manner. d. It requires large data.
- MLlib lack in some available algorithms, for example, Tanimoto distance.

Q.5 List the languages supported by Apache Spark.

Apache Spark Supports the following Languages: **Scala, Java, R, Python.**

Q.6 What are the cases where Apache Spark surpasses Hadoop?

The data processing speed increases in the **Apache Spark**. This is because of the support of *in-memory computation* by the system. Thus, the performance of the system increase by 10x-1000x times. Apache Spark uses various languages for distributed application development.

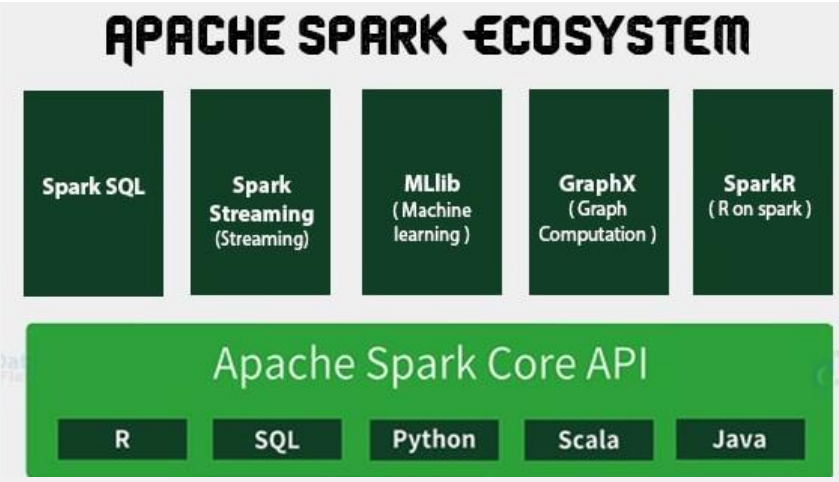
On the top of spark core, various libraries are present. These libraries enable workload that uses streaming, SQL, graph and machine learning. Some of these workloads are also supported by Hadoop. Spark facilitates the development by joining them into the same application. Apache Spark adopts micro-batching. Which is essentially used for handling near real time processing data model.

Q.7 Compare Hadoop and Spark.

- **Cost Efficient –** In Hadoop, during replication, a large number of servers, huge amount of storage, and the large data center is required. Thus, installing and using Apache Hadoop is expensive. While using Apache Spark is a cost effective solution for big data environment.

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

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



Spark Interview Questions and Answers – Spark Ecosystem

Q.8 What are the components of Spark Ecosystem?

The various components of Apache Spark are:

- **Spark Core –** Spark Core is the foundation of the whole project. All the functionality that is in Spark, is present on the top of Spark Core.
- **Spark Streaming –** It allows fault-tolerant streaming of live data streams. It is an add-on to core Spark API. Here it makes use of micro-batching for real-time streaming. Thus it packages live data into small batches and delivers to the batch system for processing.
- **Spark SQL –** Spark SQL component is distributed framework for structured data processing. Using Spark SQL Spark gets more information about the structure of data and the computation being performed. As a result, by using this information Spark can perform extra optimization.
- **Spark MLlib –** MLlib is a scalable learning library that discusses both: High-quality algorithm, High speed. The motive behind MLlib creation is to make machine learning scalable and easy. Thus. it contains machine learning libraries that have an implementation of various machine learning algorithms.
- **Spark GraphX** – GraphX is API for graphs and graph parallel execution. In order to support graph computation, graphX contains set of fundamental operators like sub graph, joinvertices and an optimized variant of Pregel API. Also, clustering,

classification, traversal, searching, and pathfinding is possible in graphX.

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

Q.9 What is Spark Core?

Spark Core is a common execution engine for Spark platform. It provides *parallel* and *distributed processing* for large data sets. All the components on the top of it. Spark core provides speed through *in-memory computation*. And for ease of development, it also supports Java, Scala and Python APIs.

RDD is the basic data structure of Spark Core. RDDs are immutable, a partitioned collection of record that can operate in parallel. We can create RDDs by transformation on existing RDDs. Also by loading an external dataset from stable storage like HDFS or **HBase**, we can form RDD.

Q.10 How is data represented in Spark?

The data can be represented in three ways in Apache Spark: *RDD, DataFrame, DataSet*.

RDD: RDD stands for *Resilient Distributed Datasets*. It is also a *Read-only* partition collection of records. RDD is the fundamental data structure of Spark. Hence, RDDs can only be created through deterministic operation on either:

- Data in stable storage.
- Parallelizing already existing collection in driver program.
- Other RDDs. RDD allows programmer perform in-memory computations on large clusters in a fault-tolerant manner. Thus, speed up the task.

DataFrame: Unlike an RDD, the data organizes into named columns, like a table in a relational database. It is also an immutable distributed collection of data. **DataFrame** allows developers to impose a structure onto a distributed collection of data, therefore, allowing higher-level abstraction.

DataSet: Dataset is an extension of DataFrame API which provides type-safe, object-oriented programming interface. **DataSet** also takes advantage

of **Spark's Catalyst optimizer** by exposing expressions and data fields to a query planner.

Q.11 What are the abstractions of Apache Spark?

The main abstraction provided by Apache Spark is **Resilient Distributed Dataset**. RDDs are fault tolerant in nature. We cannot improve the changes made in RDD. RDDs creation starts with the file in a file system like Hadoop file system and then transforming it. The shared variable is the second abstraction provided by Apache Spark. We can use this in parallel operations.

Q.12 Explain the operations of Apache Spark RDD.

Apache Spark RDD supports two types of operations: *Transformations and Actions-*

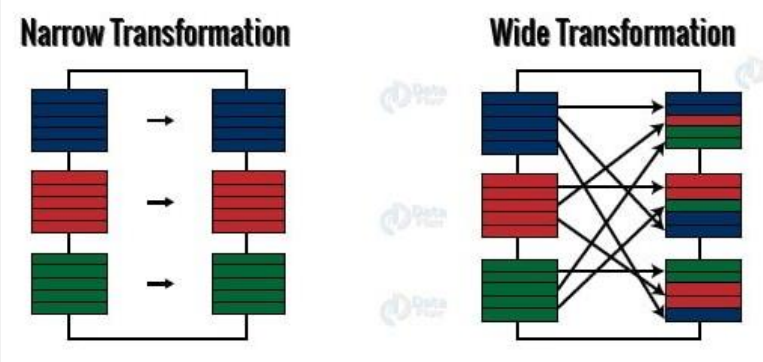
- **Transformations** are lazy operations on an RDD that create one or many new **RDDs**. For example *Map, filter, reduceByKey* etc creates new RDD. RDD create a new dataset from an existing one. That executes on demand. That means they compute lazily. Whenever we perform any transformation on RDD, it creates a new RDD each time.
- **Action** returns final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD. After application of all the intermediate transformation, it gives the final result to *driver program* or writes it out to file system. Upon applying Actions on an RDD non-RDD values gets generate.

Q.13 How many types of Transformation are there?

There are two types of transformation namely *narrow transformation* and *wide transformation*.

- **Narrow Transformation** is the result of *map, filter* and such that the data is from a single partition only. As a result, the data is self-sustained. The RDD that we get as an output has a partition with records that originate from a single partition in the parent RDD.
- **Wide transformations** are the result of *groupByKey* and *reduceByKey.* The data that we will need to

compute the records in a single partition is kept at many partitions of the parent RDD.



Apache Spark Narrow and Wide Transformations

Q.14 In how many ways RDDs can be created? Explain.
There are three ways to create an RDD:

- **Parallelized collection –** In the initial stages, the RDD is generally created by parallelized collection. In this method, we take the existing collection in the program and pass it to *parallelize()* method of *SparkContext.* The thing that should be noticed in the parallelized collection is the number of partition the dataset is cut into. For each partition of the cluster, Spark will run one task. Spark set a number of partition based on our cluster. But the number of partitions can also be set manually. Pass the number of partition as the second parameter for manual partition. e.g. sc.parallelize(data, 20), here we have manually given a number of partition as 20.
- **External Datasets (Referencing a dataset) –** In Spark one can create distributed dataset from any data source supported by Hadoop. For example the local file system, HDFS, Cassandra, HBase etc. In this, the data is loaded from the external dataset. To create text file RDD we can use **SparkContext** textFile method. It takes URL of the file and read it as a collection of line. URL can be a local path on the machine or a hdfs://, s3n://, etc.
- **Creating RDD from existing RDD –** Transformation converts one RDD into another RDD. By using transformation we can create an RDD from existing RDD. Transformation acts as a function that intakes an RDD and produces one.

Q.15 What are Paired RDD?

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

Q.16 What is meant by in-memory processing in Spark?

In **in-memory computation**, we keep data in random access memory in place of some slow disk drives. The processing of data is in parallel. Using this we can also identify the pattern, analyze large data Spark offers in in-memory capabilities. As a result, this increases the processing speed because it retrieves the data from memory in place of the disk. Also, the execution time of the process decreases. Keeping the data in-memory improves the performance by the order of magnitudes.

The main abstraction of Spark is its **RDDs**. Also, we can cache RDD using the ***cache()*** or ***persist()*** method. In *cache()* method all the RDD are in-memory. The dissimilarity between cache() and persist() is the default storage level. For cache() it is MEMORY_ONLY. While in *persist()* there are various storage levels like:

- MEMORY_ONLY,
- MEMORY_AND_DISK,
- MEMORY_ONLY_SER
- MEMORY_AND_DISK_SER
- DISK_ONLY

Q.17 How is fault tolerance achieved in Apache Spark?

The basic fault-tolerant semantic of Spark are:

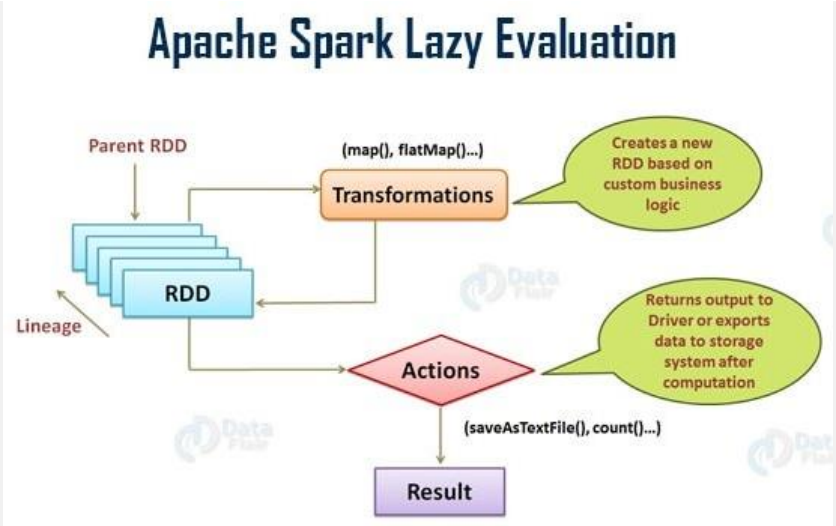
- Since all RDD is an immutable data set. Each RDD keeps track of the lineage of the deterministic operation that employee on fault-tolerant input dataset to create it.
- If any partition of an RDD is lost due to a worker node failure, then that partition can be re-computed from the original fault-tolerant dataset using the lineage of operations.
- Assuming that all of the RDD transformations are deterministic, the data in the final transformed RDD will always be the same irrespective of failures in the Spark cluster.

To achieve fault tolerance for all the generated RDDs, the achieved data replicates among multiple Spark executors in worker node in the cluster. This result in two types of data that should recover in the event of failure:

- **Data received and replicated –** In this, the data replicates on one of the other nodes. Thus we can retrieve data when a failure occurs.
- **Data received but buffered for replication –** the data does not replicate. Thus the only way to recover fault is by retrieving it again from the source.

Failure can also occur in worker and driver nodes.

- **Failure of worker node –** The node which runs the application code on the cluster is worker node. These are the slave nodes. Any of the worker nodes running executor can fail, thus resulting in loss of in-memory data. If any receivers were running on failed nodes, then their buffer data will vanish.
- **Failure of driver node –** If the driver node running the Spark Streaming application fails, then there is the loss of SparkContent. All executors along with their in-memory data vanishes.



Spark Interview Questions and Answers – Lazy Evaluation Feature

Q.18 What is Directed Acyclic Graph(DAG)?

RDDs are formed after every transformation. At high level when we apply action on these RDD, Spark creates a **DAG**. DAG is a finite directed graph with *no directed cycles*.
There are so many vertices and edges, where each edge is directed from one vertex to another. It contains a sequence of vertices such that every edge is directed from earlier to later in the sequence. It is a strict generalization of MapReduce model. DAG lets you get into the stage and expand in detail on any stage.
In the stage view, the details of all RDDs that belong to that stage are expanded.

Q.19 What is lineage graph?
Lineage graph refers to the graph that has all the parent RDDs of an RDD. It is the result of all the transformation on the RDD. It creates a logical execution plan.
A *logical execution plan* is a plan that starts with the very first RDD. Also, it does not have any dependency on any RDD. It then ends at the RDD which produces the result of an action that has been called to execute.
Q.20 What is lazy evaluation in Spark?
The **lazy evaluation** known as *call-by-need* is a strategy that delays the execution until one requires a value. The *transformation* in Spark is lazy in nature. Spark evaluate them lazily. When we call some operation in RDD it does not execute immediately; Spark maintains the graph of which operation it demands. We can execute the operation at any instance by calling the action on the data. The data does not loads until it is necessary.

Q.21 What are the benefits of lazy evaluation?
Using lazy evaluation we can:

- Increase the manageability of the program.
- Saves computation overhead and increases the speed of the system.
- Reduces the time and space complexity.
- provides the optimization by reducing the number of queries.

Q.22 What do you mean by Persistence?

RDD persistence is an optimization technique which saves the result of RDD evaluation. Using this we save the intermediate result for further use. It reduces the computation overhead. We can make persisted RDD through ***cache()*** and ***persist()*** methods. It is a key tool for the interactive algorithm. Because, when RDD is persisted each node stores any partition of it that it computes in memory. Thus makes it reusable for future use. This process speeds up the further computation ten times.

Q.23 Explain various level of persistence in Apache Spark.
The persist method allows seven storage level:

- **MEMORY_ONLY –** Store RDD as deserialized Java objects. If the RDD does not fit in memory, then some partitions will not be cached and will recompute on the fly each time needed. This is the default level.
- **MEMORY_AND_DISK –** Store RDD as deserialized Java objects. If the RDD does not fit in memory, store the partitions that don't fit on the disk, and read them from there when they're needed.
- **MEMORY_ONLY_SER (Java and Scala) –** Store RDD as serialized Java objects. This is more space-efficient than deserialized objects. especially when using a fast serializer. but it is hard for CPU to read.
- **MEMORY_AND_DISK_SER(Java and Scala) –** Like MEMORY_ONLY_SER, but spills partitions that don't fit in memory to disk.
- **DISK_ONLY –** It stores the RDD partitions only on disk.
- **MEMORY_ONLY_2, MEMORY_AND_DISK_2 –** It replicates each partition on two cluster nodes.
- **OFF_HEAP –** Like MEMORY_ONLY_SER, but store the data in off-heap memory. This requires enabling of off-heap memory.

Q.24 Explain the run-time architecture of Spark?
The components of the **run-time architecture of Spark** are as follows:
a. The driver
b. Cluster manager
c. Executors

The Driver – The *main()* method of the program runs in the driver. The process that runs the user code which creates RDDs performs transformation and action, and also creates *SparkContext* is called diver. When the Spark Shell is launched, this signifies that we have created a driver program. The application finishes, as the driver terminates. Finally, driver program splits the Spark application into the task and schedules them to run on the executor.

Cluster Manager – Spark depends on cluster manager to launch executors. In some cases, even the drivers are launched by cluster manager. It is a pluggable component in Spark. On the cluster manager, the Spark scheduler schedules the jobs and action within a spark application in FIFO fashion. Alternatively, the scheduling can also be done in Round Robin fashion. The resources used by a Spark application can also be dynamically adjusted based on the workload. Thus, the application can free unused resources and request them again when there is a demand. This is available on all coarse-grained cluster managers, i.e. **standalone mode**, **YARN mode**, and **Mesos** coarse-grained mode.

The Executors – Each task in the Spark job runs in the *Spark executors*. thus, Executors are launched once in the beginning of Spark Application and then they run for the entire lifetime of an application. Even after the failure of Spark executor, the Spark application can continue with ease. There are two main roles of the executors:

- Runs the task that makes up the application and returns the result to the driver.
- Provide in-memory storage for RDDs that the user program cache.

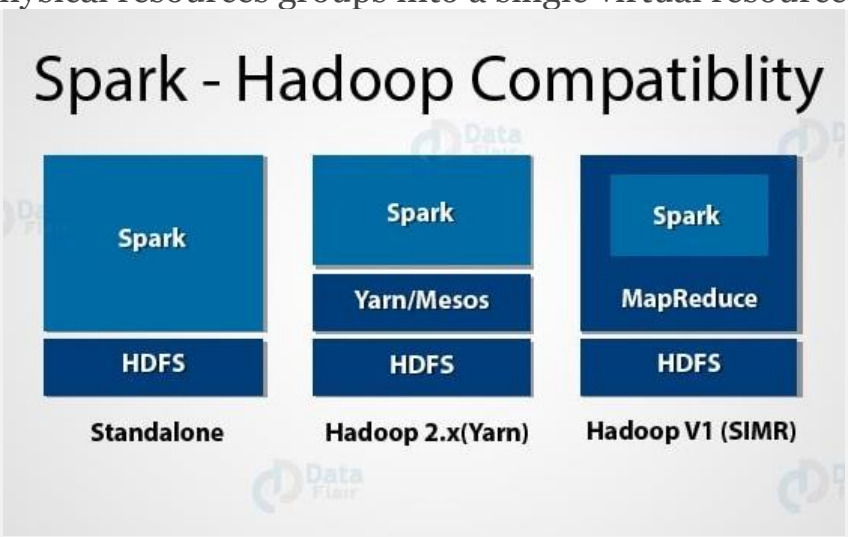
Q.25 Explain various cluster manager in Apache Spark?

The various cluster manages supported by Apache Spark are Standalone, Hadoop YARN, Apache Mesos.

- **Standalone Cluster Manager –** Standalone is a simple cluster manager of Spark that makes it easy to setup a cluster. In many cases, it is the simplest way to run Spark application in a clustered environment. It has masters and number of workers with the configured amount of memory and CPU cores. In standalone cluster mode, Spark allocates resources based on the

core. It has the constraints that only one executor can be allocated on each worker per application.

- **Hadoop YARN –** YARN became the sub-project of Hadoop in the year 2012. The key idea behind YARN is to bifurcate the functionality of resource manager and job scheduling into different daemons. The plan is to have a Global Resource Manager (RM) and per-application ***Application Master (AM)***. An application is either a **DAG** of graphs or an individual job. The data computation framework is a combination of the ResourceManager and the **NodeManager**.
- **Apache Mesos –** Apache Mesos handles the workload in a distributed environment.It is healthful for deployment and management of applications in large-scale cluster environments. Mesos clubs together the existing resource of the machines/nodes into a cluster, as a result from this a variety of workloads may utilize. This is known as node abstraction, thus it decreases an overhead of allocating a specific machine for different workloads. It is resource management platform for Hadoop and **Big Data** cluster. In some way, Apache Mesos is the reverse of virtualization. Because in virtualization one physical resource divides into multiple virtual resources, while in Mesos multiple physical resources groups into a single virtual resource.



Spark Interview Questions and Answers – Hadoop Compatibility

Read about Spark Cluster Managers in detail.

Q.26 In how many ways can we use Spark over Hadoop?

In three ways we can run Spark over Hadoop: Standalone, YARN, SIMR

- **Standalone –** In this, we can either divide resource on all machines or subset of machines in *Hadoop cluster.*
- **YARN –** We can run Spark on YARN without any pre-requisites. Thus, we can integrate Spark in Hadoop stack and take an advantage of facilities of Spark.
- **SIMR (Spark in MapReduce) –** Another way to do this is by launching Spark job inside Map reduce. With SIMR we can use Spark shell in few minutes after downloading it. This reduces the overhead of deployment, and we can play with Spark.

Q.27 What is YARN?

YARN became the sub-project of *Hadoop* in the year 2012. It is also known as MapReduce 2.0. The key idea behind YARN is to bifurcate the functionality of resource manager and job scheduling into different daemons. The plan is to have a Global *Resource Manager(RM)* and per-application *Application Master (AM)*. An application is either a **DAG** of graphs or an individual job.

The data computation framework is a combination of the ResourceManager and the NodeManager.

The **Resource Manager** manages resource among all the applications in the system. The Resource Manager has scheduled and *Application Manager.* The Scheduler allocates resource to the various running application. The Scheduler is pure Scheduler if it performs no monitoring or tracking of the status of the application. The **Application Manager** manages applications across all the nodes. NodeManager contains ApplicationMaster and container. A **container** is a place where a unit of work happens. Each task of MapReduce runs in one container. The per-application ApplicationMaster is a framework specific library. It negotiates resources from the ResourceManager and continues with **NodeManager(s)** to execute and watch the tasks. Application or job requires one or more containers. NodeManager looks after containers, resource usage (CPU, memory, disk, and network) and reporting this to the ResourceManager.

Q.28 How can we launch Spark application on YARN?

There are two deployment modes to launch Spark application on YARN: the *cluster mode* and the *client mode.*

- **In cluster mode,** the Spark driver runs inside Application Master Process and this is managed by YARN on the cluster.
- **In client mode,** the driver runs in the client process. The Application Master requests a resource from YARN. And it provides it to the driver program.

Q.29 Define Partition in Apache Spark.

Partition refers to, a logical block of large distributed Dataset. Logically partitioning the data and distributing it over the cluster provides parallelism. It also minimizes network traffic for sending data between executors. It determines how to access the entire hardware resources during job execution. RDD is automatically partitioned in Spark. We can change the size and number of the partition.

Q.30 What are shared variables?

Shared variables are one of the abstractions of Apache Spark. Shared variables can be used in parallel operations.

Whenever Spark runs a function in parallel as a set of tasks on different nodes, each variable that is used in function are circulated to each task. Sometimes there is a need to share the variables across the tasks or between the task and the driver program.

Apache Spark supports two types of shared variables namely ***broadcast variable*** and ***accumulator.***

Using broadcast variables we cache a value in memory on all nodes while we add accumulators to, such as counters and sums.

Q.31 What is Accumulator?

The *accumulator* is the type of Shared variable that is only added through associative and commutative operations. Using accumulator we can update the value of the variable while executing. We can also implement counters (as in MapReduce) or sums using an accumulator. Users can create named or unnamed accumulator. We can create numeric accumulator by calling *SparkContext.longAccumulator()* or *SparkContext.doubleAccumulator()* for Long or Double.

Q.32 What is the difference between DSM and RDD?

a) READ

- **RDD:** In RDD the read operation is *coarse grained or fine grained*. In **coarse-grained** we can transform the whole dataset but not an individual element. While in **fine-grained** we do the transformation of an individual element on a dataset.
- **Distributed Shared Memory:** The read operation in Distributed shared memory is fine-grained.

b) Write:

- **RDD:** The write operation is coarse-grained in RDD.
- **Distributed Shared Memory:** In distributed shared system the write operation is fine grained.

c) Consistency:

- **RDD:** The consistency of RDD is trivial meaning it is immutable in nature. Any changes made to an RDD cannot roll back, it is permanent. So the level of consistency is high.
- **Distributed Shared Memory:** The system guarantees that if the programmer follows the rules, the memory will be consistent. It also guarantees that the results of memory operations will be predictable.

d) Fault-recovery mechanism:

- **RDD:** Using lineage graph at any point in time we can easily find the lost data in an RDD.
- **Distributed Shared Memory:** Fault tolerance is achieved by a checkpointing technique. It allows applications to roll back to a recent checkpoint rather than restarting.

e) Straggler mitigation: Stragglers, in general, are those tasks that take more time to complete than their peers.

- **RDD:** in RDD it is possible to mitigate stragglers using backup task.
- **Distributed Shared Memory:** It is quite difficult to achieve straggler mitigation.

f) Behavior if not enough RAM:

- **RDD:** If there is not enough space to store RDD in RAM then the RDDs are shifted to disk.
- **Distributed Shared Memory:** In this type of system the performance decreases if the RAM runs out of storage.

Q.33 How can data transfer be minimized when working with Apache Spark?

By minimizing data transfer and avoiding shuffling of data we can increase the performance. In Apache Spark, we can minimize the data transfer in three ways:

- **By using a broadcast variable –** Since broadcast variable increases the efficiency of joins between small and large RDDs. the broadcast variable allows keeping a read-only variable cached on every machine in place of shipping a copy of it with tasks. We create broadcast variable v by calling *SparlContext.broadcast(v)* and we can access its value by calling the value method.
- **Using Accumulator –** Using accumulator we can update the value of a variable in parallel while executing. Accumulators can only be added through the associative and commutative operation. We can also implement counters (as in MapReduce) or sums using an accumulator. Users can create named or unnamed accumulator. We can create numeric accumulator by calling *SparkContext.longAccumulator()* or *SparkContext.doubleAccumulator()* for Long or Double respectively.
- By avoiding operations like ByKey, repartition or any other operation that trigger shuffle. we can minimize the data transfer.

Q.34 How does Apache Spark handles accumulated Metadata?

By triggering automatic cleanup Spark handles the automatic Metadata. We can trigger cleanup by setting the parameter "***spark.cleaner.ttl***". the default value for this is infinite. It tells for how much duration Spark will remember the metadata. It is periodic cleaner. And also ensure that metadata older than the set duration will vanish. Thus, with its help, we can run Spark for many hours.

Q.35 What are the common faults of the developer while using Apache Spark?

The common mistake by developers are:

- Customer hit web-service several time by using multiple clusters.
- Customer runs everything on local node instead of distributing it.

Q.36 Which among the two is preferable for the project- Hadoop MapReduce or Apache Spark?
The answer to this question depends on the type of project one has. As we all know Spark makes use of a large amount of RAM and also needs a dedicated machine to provide an effective result. Thus the answer depends on the project and the budget of the organization.

Q.37 List the popular use cases of Apache Spark.
The most popular use-cases of Apache Spark are:
1. Streaming
2. Machine Learning
3. interactive Analysis
4. fog computing
5. Using Spark in the real world

Q.38 What is Spark.executor.memory in a Spark Application?
The default value for this is 1 GB. It refers to the amount of memory that will be used per executor process.
We have categorized the above Spark Interview Questions and Answers for Freshers and Experienced-

- Spark Interview Questions and Answers for Fresher – Q.No.1-8, 37
- Spark Interview Questions and Answers for Experienced – Q.No. 9-36, 38

b. Spark SQL Interview Questions and Answers

In this section, we will discuss some basic Spark SQL Interview Questions and Answers.

Q.39 What is DataFrames?
It is a collection of data which organize in named columns. It is theoretically equivalent to a table in relational database. But it is more optimized. Just like RDD, DataFrames evaluates lazily. Using lazy evaluation we can optimize the execution. It optimizes by applying the techniques such as bytecode generation and predicate push-downs.

Q.40 What are the advantages of DataFrame?

1. It makes large data set processing even easier. Data Frame also allows developers to impose a structure onto a distributed collection of data. As a result, it allows higher-level abstraction.
2. Data frame is both space and performance efficient.
3. It can deal with both structured and unstructured data formats, for example, Avro, CSV etc . And also storage systems like HDFS, HIVE tables, MySQL, etc.
4. The DataFrame API's are available in various programming languages. For example Java, Scala, Python, and R.
5. It provides Hive compatibility. As a result, we can run unmodified Hive queries on existing Hive warehouse.
6. Catalyst tree transformation uses DataFrame in four phases: a) Analyze logical plan to solve references. b) Logical plan optimization c) Physical planning d) Code generation to compile part of the query to Java bytecode.
7. It can scale from kilobytes of data on the single laptop to petabytes of data on the large cluster.

Q.41 What is DataSet?

Spark Datasets are the extension of Dataframe API. It creates object-oriented programming interface and type-safety. Dataset is Spark 1.6 release. It makes use of Spark's catalyst optimizer. It reveals expressions and data fields to a query optimizer. Dataset also influences fast in-memory encoding. It also provides provision for compile time type-safety. We can check for errors in an application when they run.

Q.42 What are the advantages of DataSets?

- It provides run-time type safety.
- Influences fast in-memory encoding.
- It provides a custom view of structured and semi-structured data.
- It owns rich semantics and an easy set of domain-specific operations, as a result, it facilitates the use of structured data.
- Dataset API decreases the use of memory. As Spark knows the structure of data in the dataset, thus it creates an optimal layout in memory while caching.

Q.43 Explain Catalyst framework.

The **Catalyst** is a framework which represents and manipulate a DataFrame graph. Data flow graph is a tree of relational operator and expressions. The three main features of catalyst are:

- It has a TreeNode library for transforming tree. They are expressed as Scala case classes.
- A logical plan representation for relational operator.
- Expression library.

The *TreeNode* builds a query optimizer. It contains a number of the query optimizer. **Catalyst Optimizer** supports both *rule-based* and *cost-based optimization*. In rule-based optimization the optimizer use set of rule to determine how to execute the query. While the cost based optimization finds the most suitable way to carry out SQL statement. In cost-based optimization, many plans are generates using rules. And after this, it computes their cost. Catalyst optimizer makes use of standard features of Scala programming like pattern matching.

Q.44 List the advantage of Parquet files.

- It is efficient for large scale queries.
- It supports various efficient compression and encoding Scheme.
- It consumes less space.

We have categorized the above frequently asked Spark Interview Questions and Answers for Freshers and Experienced-

- **Spark Interview Questions and Answers for Fresher – Q.No. 39-42**
- **Spark Interview Questions and Answers for Experienced – Q.No. 43, 44**

c. Spark Streaming Interview Questions and Answers

In this section, we will discuss some basic Spark Interview Questions and Answers based on Spark Streaming.

Q.45 What is Spark Streaming?

Through ***Spark streaming***, we achieve fault tolerant processing of live data stream. The input data can be from any source. For example, like *Kafka,*

Flume, kinesis, twitter or HDFS/S3. It gives the data to filesystems, databases, and live dashboards after processing. The working of Spark Streaming is as under:

- Spark Streaming takes in the live data.
- The input data stream is divided into batches of input data.
- The Spark engine processes the batches of input data. The final result is also in batches.

Q.46 What is DStream?

DStream is the high-level abstraction provided by *Spark Streaming*. It represents a continuous stream of data. Thus, DStream is internally a sequence of RDDs. There are two ways to create DStream:

- by using data from different sources such as Kafka, Flume, and Kinesis.
- by applying high-level operations on other DStreams.

Q.47 Explain different transformation on DStream.

DStream is a basic abstraction of Spark Streaming. It is a continuous sequence of RDD which represents a continuous stream of data. Like RDD, DStream also supports many transformations which are available on normal Spark RDD. For example*, map(func), flatMap(func), filter(func)* etc.

Q.48 Does Apache Spark provide checkpointing?

Yes, Apache Spark provides checkpointing. Apache supports two types of checkpointing:

- **Reliable Checkpointing:** It refers to that checkpointing in which the actual RDD is saved in the 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.

Q.49 What is write ahead log(journaling)?

The ***write-ahead log*** is a technique that provides durability in a database system. It works in the way that all the operation that applies on data, we write it to write-ahead log. The logs are durable in nature. Thus, when the failure occurs we can easily recover the data from these logs. When we enable the write-ahead log Spark stores the data in fault-tolerant file system.

Q.50 What is a reliable and unreliable receiver in Spark?

- **Reliable Receiver –** A reliable receiver acknowledges to the source on receiving data and stores it. Implementing this receiver involves examining of the semantics of source acknowledgments.
- **Unreliable Receiver –** An unreliable receiver does not send an acknowledgment to a source. It is for sources that do not bear an acknowledgment. It is also for reliable sources when one does not want to enter the complexity of acknowledgment.

We have categorized the above Spark Interview Questions and Answers for Freshers and Experienced-

- **Spark Interview Questions and Answers for Fresher – Q.No. 45-47**
- **Spark Interview Questions and Answers for Experienced – Q.No. 48-50**

d. Spark MLlib Interview Questions and Answers

Here are some Spark Interview Questions and Answers for freshers and experienced based on MLlib.

Q.51 What is Spark MLlib?

MLlib is the name of *Spark's machine learning library*. The various tools provided by MLlib are:

- **ML Algorithms:** It contains common learning algorithms such as classification, regression, etc.
- **Featurization:** it has tools for feature extraction, dimensionality reduction, and selection. It also has tools for constructing, evaluating and tuning ML Pipelines.

Q.52 What is Sparse Vector?

A local vector contains both the integer type and 0-based indices. It also has double-typed values, which is stored on a single machine. In ***MLlib*** two types of local vectors are a supportive namely Dense and Sparse vector. The sparse vector is one in which most of the entries are zero.

Q.53 How to create a Sparse vector from a dense vector?

Vector sparseVector = Vectors.sparse(4, new int[] {1, 3}, new double[] {3.0, 4.0});