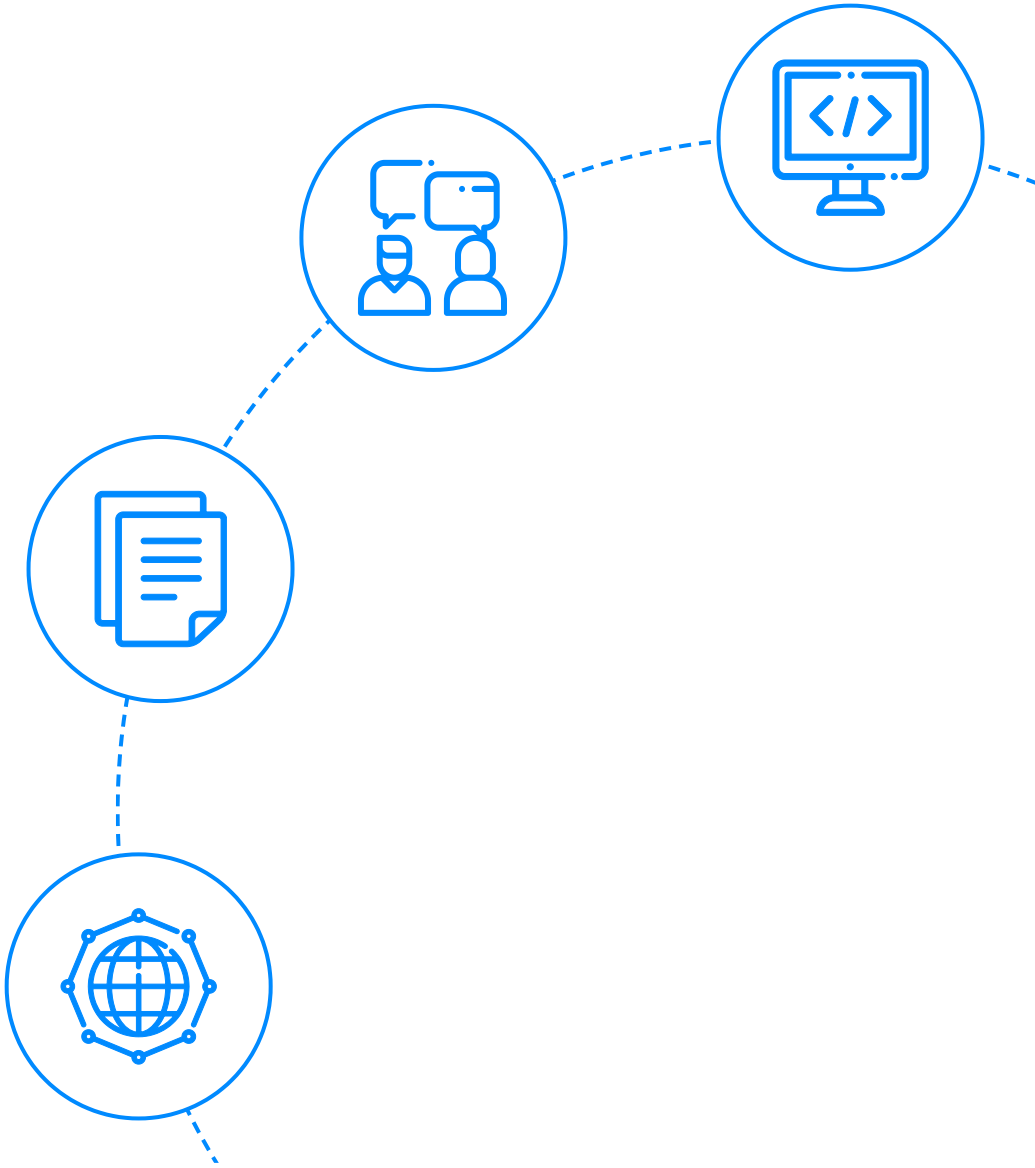
InterviewBit

Spark Interview Questions



To view the live version of the page, click here.

Contents

Spark Interview Questions for Freshers

Spark Interview Questions for Experienced

Spark Interview Questions for Experienced

(.....Continued)

19. Define Executor Memory in Spark
20. What are the functions of SparkCore?
21. What do you understand by worker node?
22. What are some of the demerits of using Spark in applications?
23. How can the data transfers be minimized while working with Spark?
24. What is SchemaRDD in Spark RDD?
25. What module is used for implementing SQL in Apache Spark?
26. What are the different persistence levels in Apache Spark?
27. What are the steps to calculate the executor memory?
28. Why do we need broadcast variables in Spark?
29. Differentiate between Spark Datasets, Dataframes and RDDs.
30. Can Apache Spark be used along with Hadoop? If yes, then how?
31. What are Sparse Vectors? How are they different from dense vectors?
32. How are automatic clean-ups triggered in Spark for handling the accumulated metadata?
33. How is Caching relevant in Spark Streaming?
34. Define Piping in Spark.
35. What API is used for Graph Implementation in Spark?
36. How can you achieve machine learning in Spark?

Conclusion

Conclusion

(.....Continued)

37. Conclusion

Let's get Started

Apache Spark is an open-source, lightning-fast computation technology build based on Hadoop and MapReduce technologies that support various computational techniques for fast and efficient processing. Spark is known for its in-memory cluster computation that is the main contributing feature for increasing the processing speed of the spark applications. Spark was developed as part of Hadoop's subproject by Matei Zaharia in 2009 at UC Berkeley's AMPLab. It was later open-sourced in the year 2010 under the BSD License which was then donated to the Apache Software Foundation in the year 2013. From 2014 onwards, Spark grabbed the top-level position among all the projects undertaken by Apache Foundation.

This article covers the most commonly asked interview questions in Spark technology and helps the software engineers and the data engineers to equip themselves for the interview. The questions range from basic to intermediate to advanced levels based on the Spark concepts.

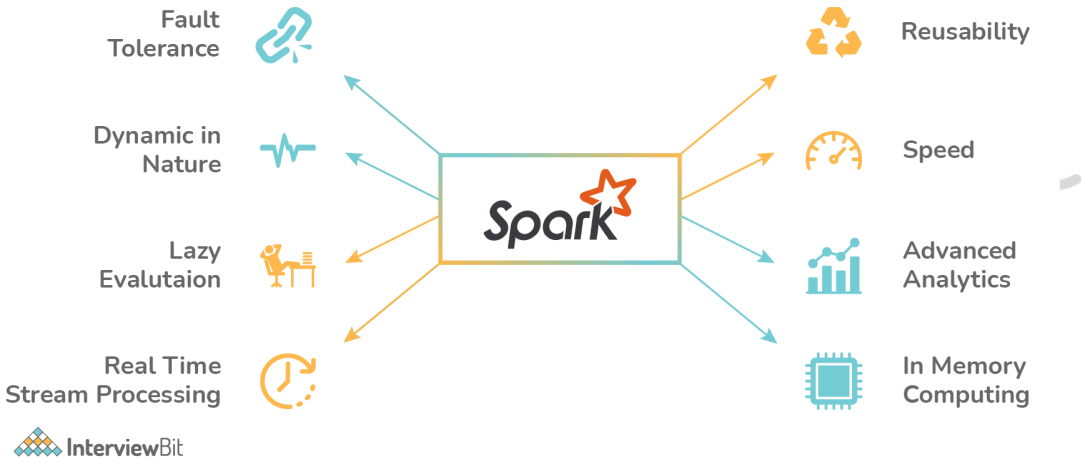
Spark Interview Questions for Freshers

1. Can you tell me what is Apache Spark about?

Apache Spark is an open-source framework engine that is known for its speed, easy-to-use nature in the field of big data processing and analysis. It also has built-in modules for graph processing, machine learning, streaming, SQL, etc. The spark execution engine supports in-memory computation and cyclic data flow and it can run either on cluster mode or standalone mode and can access diverse data sources like HBase, HDFS, Cassandra, etc.

2. What are the features of Apache Spark?

Features of 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**: Spark transformations done using Spark RDDs are lazy. Meaning, they do not generate results right away, but they create new RDDs from existing RDD. This lazy evaluation increases the system efficiency.
- **Support Multiple Languages**: Spark supports multiple languages like R, Scala, Python, Java which provides dynamicity and helps in overcoming the Hadoop limitation of application development only using Java.
- **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.

3. What is RDD?

RDD stands for Resilient Distribution Datasets. It is a fault-tolerant collection of parallel running operational elements. The partitioned data of RDD is distributed and immutable. There are two types of datasets:

- **Parallelized collections**: Meant for running parallelly.
- **Hadoop datasets**: These perform operations on file record systems on HDFS or other storage systems.

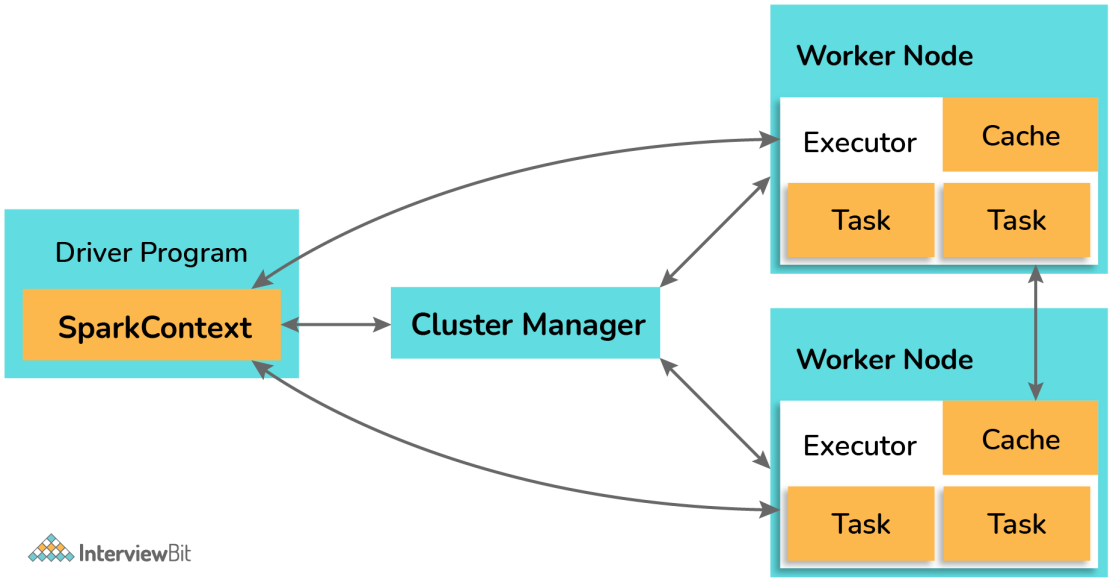
4. What does DAG refer to in Apache Spark?

DAG stands for Directed Acyclic Graph with no directed cycles. There would be finite vertices and edges. Each edge from one vertex is directed to another vertex in a *sequential manner*. The vertices refer to the RDDs of Spark and the edges represent the operations to be performed on those RDDs.

5. List the types of Deploy Modes in Spark.

There are 2 deploy modes in Spark. They are:

- **Client Mode**: The deploy mode is said to be in client mode when the spark driver component runs on the machine node from where the spark job is submitted.
  - The main disadvantage of this mode is if the machine node fails, then the entire job fails.
  - This mode supports both interactive shells or the job submission commands.
  - The performance of this mode is worst and is not preferred in production environments.
- **Cluster Mode**: If the spark job driver component does not run on the machine from which the spark job has been submitted, then the deploy mode is said to be in cluster mode.
  - The spark job launches the driver component within the cluster as a part of the sub-process of ApplicationMaster.
  - This mode supports deployment only using the spark-submit command (interactive shell mode is not supported).
  - Here, since the driver programs are run in ApplicationMaster, in case the program fails, the driver program is re-instantiated.
  - In this mode, there is a dedicated cluster manager (such as stand-alone, YARN, Apache Mesos, Kubernetes, etc) for allocating the resources required for the job to run as shown in the below architecture.



Apart from the above two modes, if we have to run the application on our local machines for unit testing and development, the deployment mode is called "**Local Mode**". Here, the jobs run on a single JVM in a single machine which makes it highly inefficient as at some point or the other there would be a shortage of resources which results in the failure of jobs. It is also not possible to scale up resources in this mode due to the restricted memory and space.

6. 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. They are created by using streaming contexts in the form of long-running tasks that are scheduled for operating in a round-robin fashion. 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 acknowledegment 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.

7. What is the difference between repartition and coalesce?

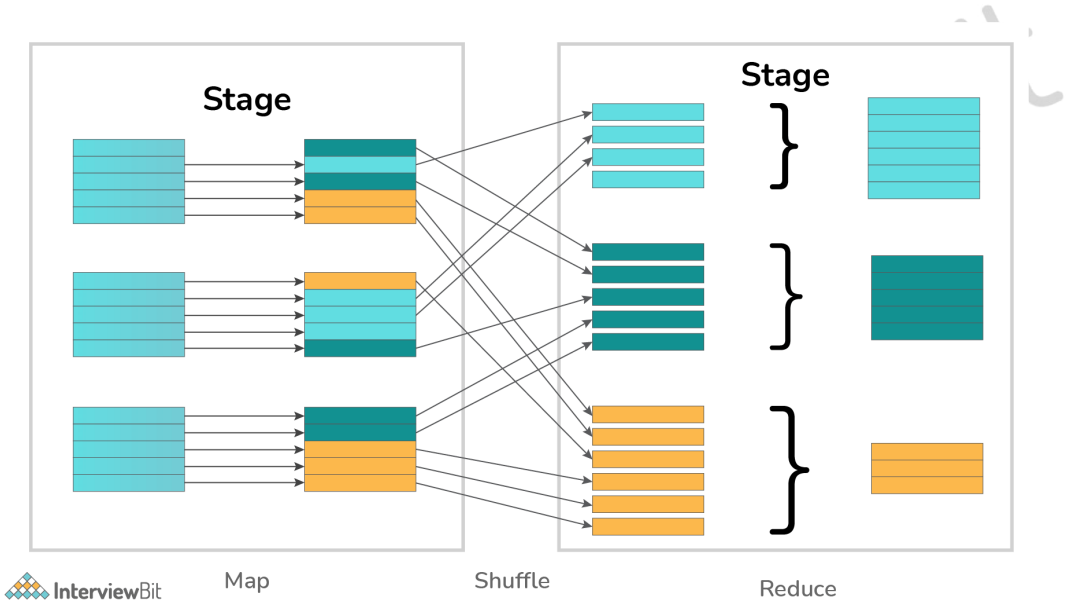
Repartition	Coalesce
Usage repartition can increase/decrease the number of data partitions.	Spark coalesce can only reduce the number of data partitions.
Repartition creates new data partitions and performs a full shuffle of evenly distributed data.	Coalesce makes use of already existing partitions to reduce the amount of shuffled data unevenly.
Repartition internally calls coalesce with shuffle parameter thereby making it slower than coalesce.	Coalesce is faster than repartition. However, if there are unequal-sized data partitions, the speed might be slightly slower.

8. 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.

9. What do you understand by Shuffling in Spark?

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.

10. What is YARN in Spark?

- YARN is one of the key features provided by Spark that provides a central resource management platform for delivering scalable operations throughout the cluster.
- YARN is a cluster management technology and a Spark is a tool for data processing.

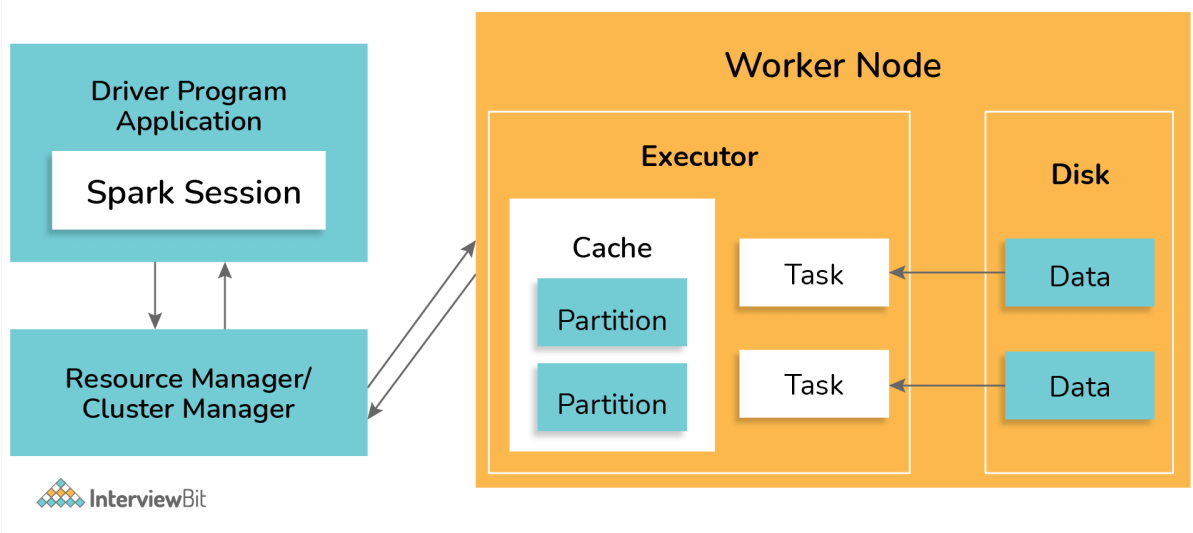
Spark Interview Questions for Experienced

11. How is Apache Spark different from MapReduce?

MapReduce	Apache Spark
MapReduce does only batch-wise processing of data.	Apache Spark can process the data both in real-time and in batches.
MapReduce does slow processing of large data.	Apache Spark runs approximately 100 times faster than MapReduce for big data processing.
MapReduce stores data in HDFS (Hadoop Distributed File System) which makes it take a long time to get the data.	Spark stores data in memory (RAM) which makes it easier and faster to retrieve data when needed.
MapReduce highly depends on disk which makes it to be a high latency framework.	Spark supports in-memory data storage and caching and makes it a low latency computation framework.
MapReduce requires an external scheduler for jobs.	Spark has its own job scheduler due to the in-memory data computation.

12. Explain the working of Spark with the help of its architecture.

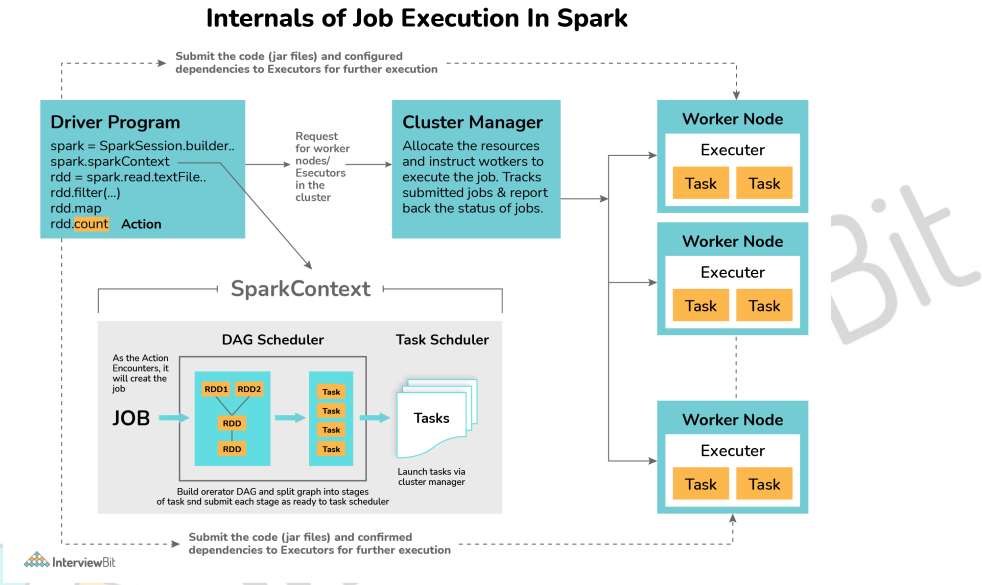
Spark applications are run in the form of independent processes that are well coordinated by the Driver program by means of a SparkSession object. The cluster manager or the resource manager entity of Spark assigns the tasks of running the Spark jobs to the worker nodes as per one task per partition principle. There are various iterations algorithms that are repeatedly applied to the data to cache the datasets across various iterations. Every task applies its unit of operations to the dataset within its partition and results in the new partitioned dataset. These results are sent back to the main driver application for further processing or to store the data on the disk. The following diagram illustrates this working as described above:



13. 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. The DAG created is submitted to the DAG Scheduler which splits the graphs into stages of tasks based on the transformations applied to the data. The stage view has the details of the RDDs of that stage.

The working of DAG in spark is defined as per the workflow diagram below:



- The first task is to interpret the code with the help of an interpreter. If you use the Scala code, then the Scala interpreter interprets the code.
- Spark then creates an operator graph when the code is entered in the Spark console.
- When the action is called on Spark RDD, the operator graph is submitted to the DAG Scheduler.
- The operators are divided into stages of task by the DAG Scheduler. The stage consists of detailed step-by-step operation on the input data. The operators are then pipelined together.
- The stages are then passed to the Task Scheduler which launches the task via the cluster manager to work on independently without the dependencies between the stages.
- The worker nodes then execute the task.

Each RDD keeps track of the pointer to one/more parent RDD along with its relationship with the parent. For example, consider the operation `val childB=parentA.map()` on RDD, then we have the RDD childB that keeps track of its parentA which is called **RDD lineage**.

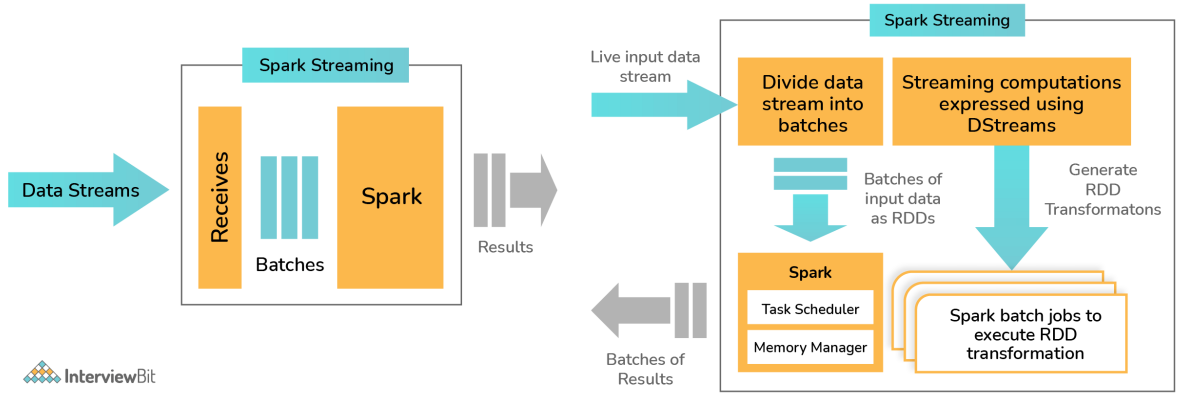
14. Under what scenarios do you use Client and Cluster modes for deployment?

- 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.

15. 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, Kinesis, 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.



16. Write a spark program to check if a given keyword exists in a huge text file or not?

```
def keywordExists(line):
   if (line.find("my_keyword") > -1):
       return 1
   return 0
lines = sparkContext.textFile("test_file.txt");
isExist = lines.map(keywordExists);
sum = isExist.reduce(sum);
print("Found" if sum>0 else "Not Found")
```

17. What can you say about Spark Datasets?

Spark Datasets are those data structures of SparkSQL that provide JVM objects with all the benefits (such as data manipulation using lambda functions) of RDDs alongside Spark SQL-optimised execution engine. This was introduced as part of Spark since version 1.6.

- Spark datasets are strongly typed structures that represent the structured queries along with their encoders.
- They provide type safety to the data and also give an object-oriented programming interface.
- The datasets are more structured and have the lazy query expression which helps in triggering the action. Datasets have the combined powers of both RDD and Dataframes. Internally, each dataset symbolizes a logical plan which informs the computational query about the need for data production. Once the logical plan is analyzed and resolved, then the physical query plan is formed that does the actual query execution.

Datasets have the following features:

- **Optimized Query feature**: Spark datasets provide optimized queries using Tungsten and Catalyst Query Optimizer frameworks. The Catalyst Query Optimizer represents and manipulates a data flow graph (graph of expressions and relational operators). The Tungsten improves and optimizes the speed of execution of Spark job by emphasizing the hardware architecture of the Spark execution platform.
- **Compile-Time Analysis**: Datasets have the flexibility of analyzing and checking the syntaxes at the compile-time which is not technically possible in RDDs or Dataframes or the regular SQL queries.
- **Interconvertible**: The type-safe feature of datasets can be converted to "untyped" Dataframes by making use of the following methods provided by the Datasetholder:
  - toDS():Dataset[T]
  - toDF():DataFrame
  - toDF(columnName:String*):DataFrame
- **Faster Computation:** Datasets implementation are much faster than those of the RDDs which helps in increasing the system performance.
- **Persistent storage qualified**: Since the datasets are both queryable and serializable, they can be easily stored in any persistent storages.
- **Less Memory Consumed**: Spark uses the feature of caching to create a more optimal data layout. Hence, less memory is consumed.
- **Single Interface Multiple Languages**: Single API is provided for both Java and Scala languages. These are widely used languages for using Apache Spark. This results in a lesser burden of using libraries for different types of inputs.

18. Define Spark DataFrames.

Spark Dataframes are the distributed collection of datasets organized into columns similar to SQL. It is equivalent to a table in the relational database and is mainly optimized for big data operations.
Dataframes can be created from an array of data from different data sources such as external databases, existing RDDs, Hive Tables, etc. Following are the features of Spark Dataframes:

- Spark Dataframes have the ability of processing data in sizes ranging from Kilobytes to Petabytes on a single node to large clusters.
- They support different data formats like CSV, Avro, elastic search, etc, and various storage systems like HDFS, Cassandra, MySQL, etc.
- By making use of SparkSQL catalyst optimizer, state of art optimization is achieved.
- It is possible to easily integrate Spark Dataframes with major Big Data tools using SparkCore.

19. Define Executor Memory in Spark

The applications developed in Spark have the same fixed cores count and fixed heap size defined for spark executors. The 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. 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.

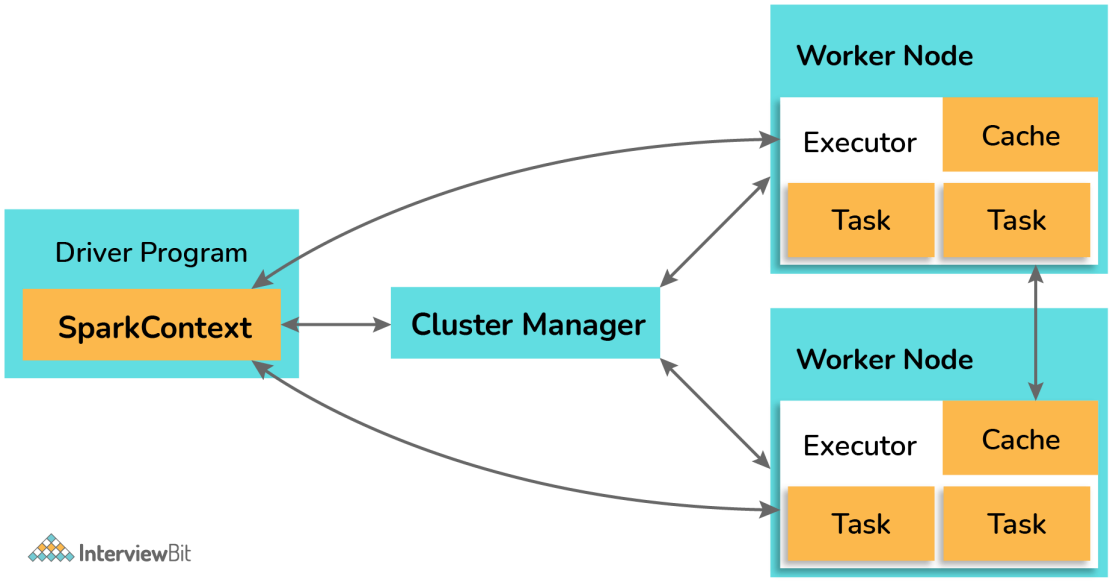
20. What are the functions of SparkCore?

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.
Spark Core does important functions such as memory management, job monitoring, fault-tolerance, storage system interactions, job scheduling, and providing support for all the basic I/O functionalities. There are various additional libraries built on top of Spark Core which allows diverse workloads for SQL, streaming, and machine learning. They are responsible for:

- Fault recovery
- Memory management and Storage system interactions
- Job monitoring, scheduling, and distribution
- Basic I/O functions

21. 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.



22. 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.

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

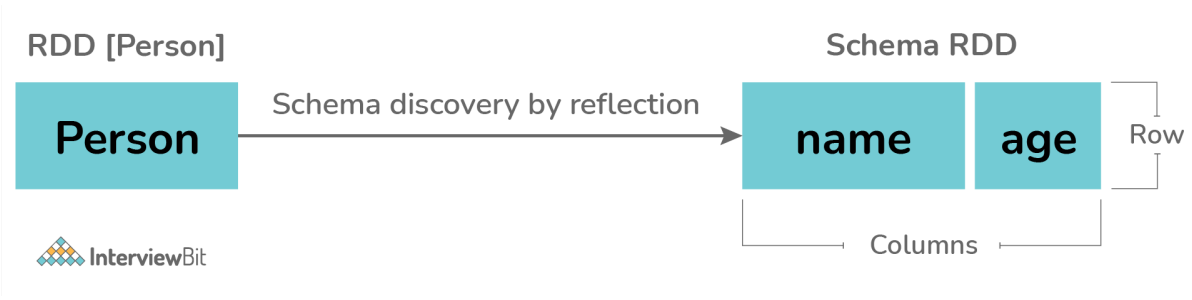
Data transfers correspond to the process of shuffling. Minimizing these transfers results in faster and reliable running Spark applications. There are various ways in which these can be minimized. They are:

- 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.

24. 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.



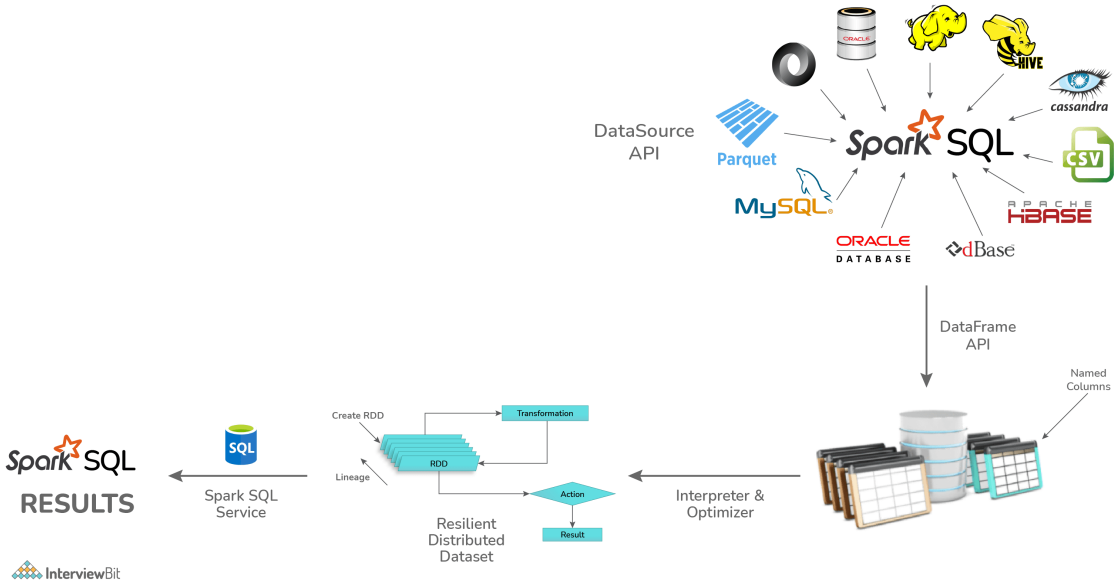
25. What module is used for implementing SQL in Apache Spark?

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.



26. 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>)
```

The following table summarizes the details of persistence levels:

Persistence Level	Space Consumed	CPU time	In
MEMORY_ONLY	High	Low	Ye
MEMORY_ONLY_SER	Low	High	Ye
MEMORY_AND_DISK	High	Medium	Sc
MEMORY_AND_DISK_SER	Low	High	Sc
DISK_ONLY	Low	High	No
OFF_HEAP	Low	High	Ye he

27. What are the steps to calculate the executor memory?

Consider you have the below details regarding the cluster:

```
Number of nodes = 10
Number of cores in each node = 15 cores
RAM of each node = 61GB
```

To identify the number of cores, we follow the approach:

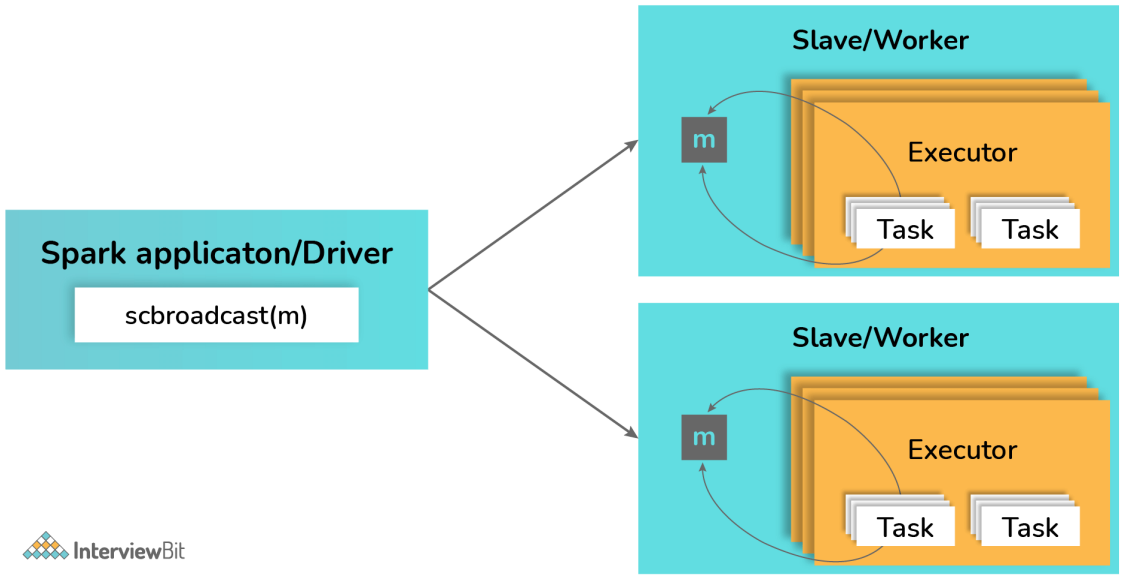
```
Number of Cores = number of concurrent tasks that can be run parallelly by the executor
```

Hence to calculate the number of executors, we follow the below approach:

```
Number of executors = Number of cores/Concurrent Task
                    = 15/5
                    = 3
Number of executors = Number of nodes * Number of executor in each node
                    = 10 * 3
                    = 30 executors per Spark job
```

28. Why do we need broadcast variables in Spark?

Broadcast variables let the developers maintain read-only variables cached on each machine instead of shipping a copy of it with tasks. They are used to give every node copy of a large input dataset efficiently. These variables are broadcasted to the nodes using different algorithms to reduce the cost of communication.

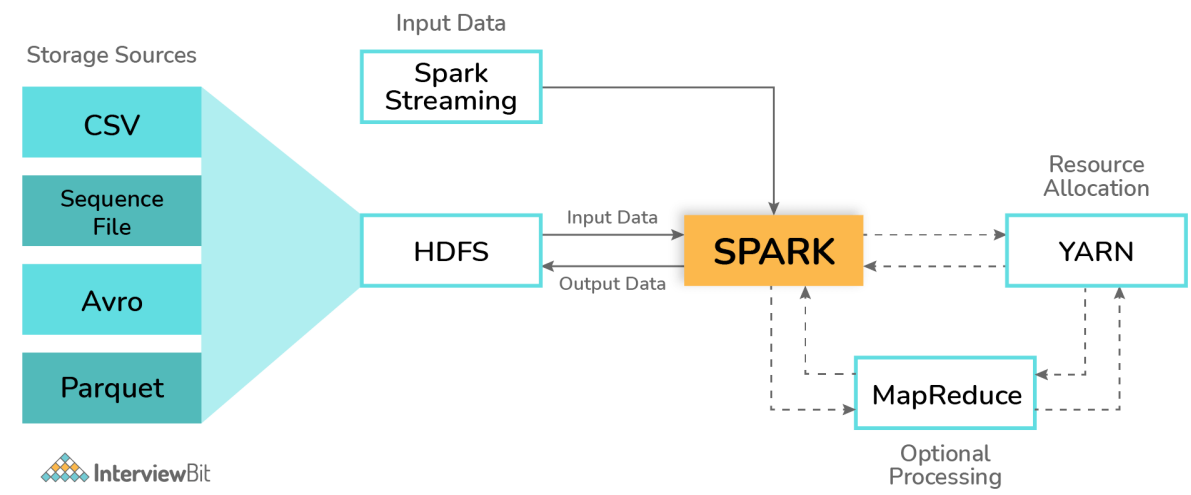


29. Differentiate between Spark Datasets, Dataframes and RDDs.

Criteria	Spark Datasets	Spark Dataframes	Spa
Representation of Data	Spark Datasets is a combination of Dataframes and RDDs with features like static type safety and object-oriented interfaces.	Spark Dataframe is a distributed collection of data that is organized into named columns.	Spa are dist coll data sche
Optimization	Datasets make use of catalyst optimizers for optimization.	Dataframes also makes use of catalyst optimizer for optimization.	The buil opti engi
Schema Projection	Datasets find out schema automatically using SQL Engine.	Dataframes also find the schema automatically.	Sch nee defi man RDD
Aggregation Speed	Dataset aggregation is faster than RDD but slower than Dataframes.	Aggregations are faster in Dataframes due to the provision of easy and powerful APIs.	RDD slow both Data and Data whil perf ever ope like grou

30. Can Apache Spark be used along with Hadoop? If yes, then how?

Yes! The main feature of Spark is its compatibility with Hadoop. This makes it a powerful framework as using the combination of these two helps to leverage the processing capacity of Spark by making use of the best of Hadoop's YARN and HDFS features.



Hadoop can be integrated with Spark in the following ways:

- **HDFS**: Spark can be configured to run atop HDFS to leverage the feature of distributed replicated storage.
- **MapReduce**: Spark can also be configured to run alongside the MapReduce in the same or different processing framework or Hadoop cluster. Spark and MapReduce can be used together to perform real-time and batch processing respectively.
- **YARN**: Spark applications can be configured to run on YARN which acts as the cluster management framework.

31. 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.

32. 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.

33. How is Caching relevant in Spark Streaming?

Spark Streaming involves the division of data stream's data into batches of X seconds called DStreams. These DStreams let the developers cache the data into the memory which can be very useful in case the data of DStream is used for multiple computations. The caching of data can be done using the cache() method or using persist() method by using appropriate persistence levels. The default persistence level value for input streams receiving data over the networks such as Kafka, Flume, etc is set to achieve data replication on 2 nodes to accomplish fault tolerance.

- Caching using cache method:

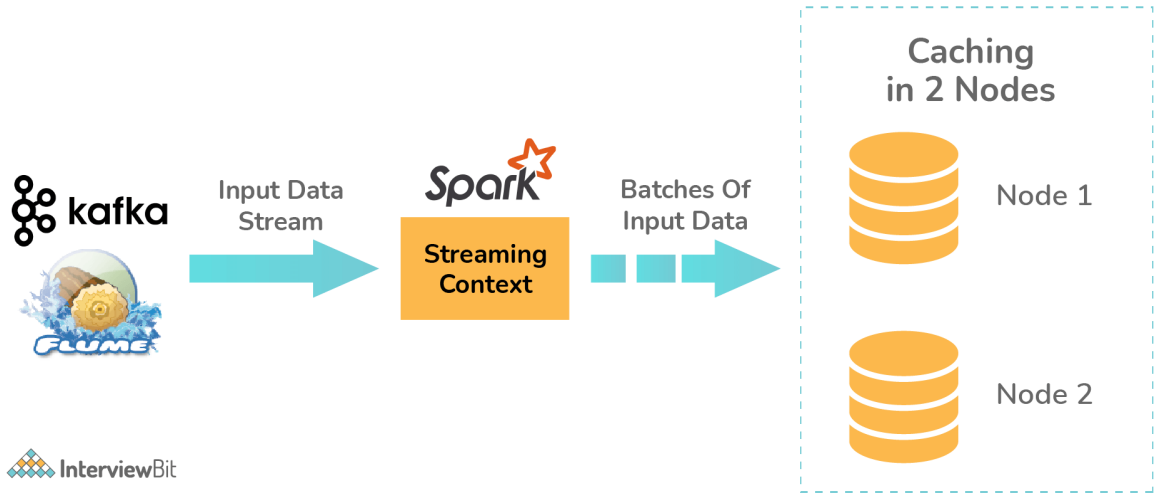
```
val cacheDf = dframe.cache()
```

- 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.



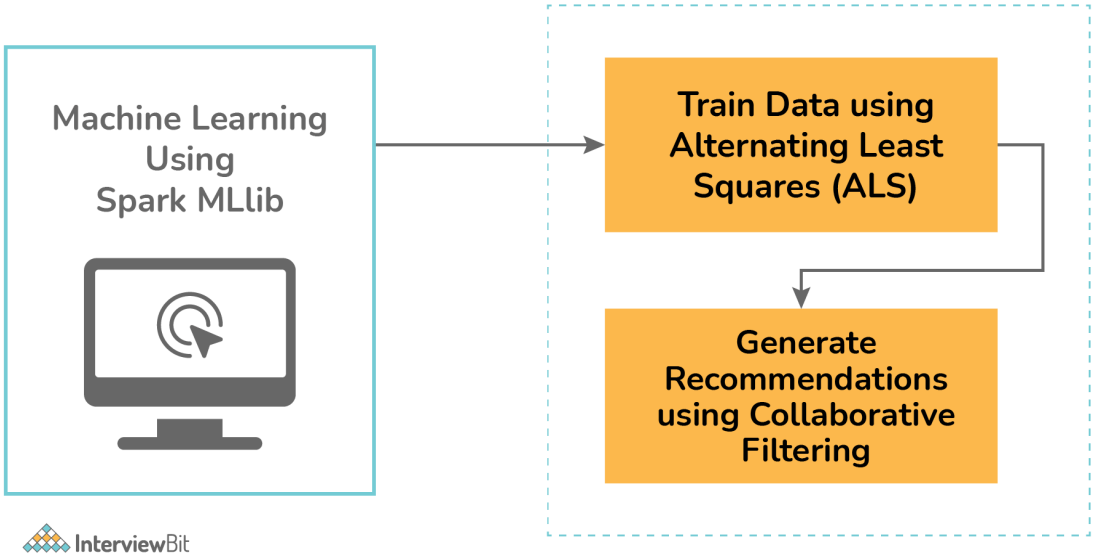
34. Define Piping in Spark.

35. 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.

36. 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. More information about this library can be obtained in detail from Spark's official documentation site here: https://spark.apache.org/docs/latest/ml-guide.html



Conclusion

Useful Resources:

https://spark.apache.org/docs/latest/

https://sparkbyexamples.com/

Links to More Interview Questions

C Interview Questions

Php Interview Questions

C Sharp Interview Questions

Web Api Interview Questions

Hibernate Interview Questions

Node Js Interview Questions

Cpp Interview Questions

Oops Interview Questions

Devops Interview Questions

Machine Learning Interview Questions

Docker Interview Questions

Mysql Interview Questions

Css Interview Questions

Laravel Interview Questions

Asp Net Interview Questions

Django Interview Questions

Dot Net Interview Questions

Kubernetes Interview Questions

Operating System Interview Questions

React Native Interview Questions

Aws Interview Questions

Git Interview Questions

Java 8 Interview Questions

Mongodb Interview Questions

Dbms Interview Questions

Spring Boot Interview Questions

Power Bi Interview Questions

Pl Sql Interview Questions

Tableau Interview Questions

Linux Interview Questions

Ansible Interview Questions

Java Interview Questions

Jenkins Interview Questions