Apache Spark Interview Questions And Answers

1. Compare Hadoop and Spark.

We will compare Hadoop MapReduce and Spark based on the following aspects:

Apache Spark vs. Hadoop		
Feature Criteria	**Apache Spark**	**Hadoop**
Speed	100 times faster than Hadoop	Decent speed
Processing	Real-time & Batch processing	Batch processing only
Difficulty	Easy because of high level modules	Tough to learn
Recovery	Allows recovery of partitions	Fault-tolerant
Interactivity	Has interactive modes	No interactive mode except Pig & Hive

Table: *Apache Spark versus Hadoop*

Let us understand the same using an interesting analogy.

"Single cook cooking an entree is regular computing. Hadoop is multiple cooks cooking an entree into pieces and letting each cook her piece. Each cook has a separate stove and a food shelf. The first cook cooks the meat, the second cook cooks the sauce. This phase is called "Map". A the end the main cook assembles the complete entree. This is called "Reduce". For Hadoop, the cooks are not allowed to keep things on the stove between operations. Each time you make a particular operation, the cook puts results on the shelf. This slows things down. For Spark, the cooks are allowed to keep things on the stove between operations. This speeds things up. Finally, for Hadoop the recipes are written in a language which is illogical and hard to understand. For Spark, the recipes are nicely written." – Stan Kladko, Galactic Exchange.io

2. What is Apache Spark?

- ***Apache Spark*** is an open-source cluster computing framework for real-time processing.
- It has a thriving open-source community and is the most active Apache project at the moment.
- Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

Figure: *Real Time Processing In Spark*

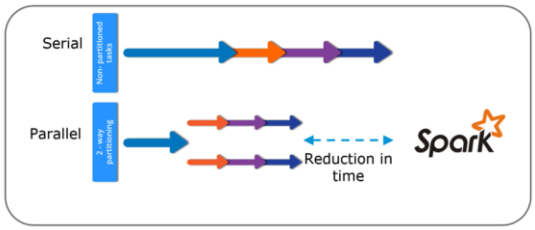


Figure: *Data Parallelism In Spark*

Spark is of the most successful projects in the Apache Software Foundation. Spark has clearly evolved as the market leader for Big Data processing. Many organizations run Spark on clusters with thousands of nodes. Today, Spark is being adopted by major players like Amazon, eBay, and Yahoo!

3. Explain the key features of Apache Spark.

The following are the key features of Apache Spark:

1. **Polyglot**
2. **Speed**
3. **Multiple Format Support**
4. **Lazy Evaluation**
5. **Real Time Computation**
6. **Hadoop Integration**
7. **Machine Learning**

Let us look at these features in detail:

1. **Polyglot**: Spark provides high-level APIs in Java, Scala, Python and R. Spark code can be written in any of these four languages. It provides a shell in Scala and Python. The Scala shell can be accessed through **./bin/spark-shell** and Python shell through **./bin/pyspark** from the installed directory.
2. **Speed**: Spark runs upto 100 times faster than Hadoop MapReduce for large-scale data processing. Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.
3. **Multiple Formats**: Spark supports multiple data sources such as Parquet, JSON, Hive and Cassandra. The Data Sources API provides a pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.
4. **Lazy Evaluation**: Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For transformations, Spark adds them to a DAG of computation and only when the driver requests some data, does this DAG actually gets executed.

5. **Real Time Computation**: Spark's computation is real-time and has less latency because of its in-memory computation. Spark is designed for massive scalability and the Spark team has documented users of the system running production clusters with thousands of nodes and supports several computational models.
6. **Hadoop Integration**: Apache Spark provides smooth compatibility with Hadoop. This is a great boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.
7. **Machine Learning**: Spark's MLlib is the machine learning component which is handy when it comes to big data processing. It eradicates the need to use multiple tools, one for processing and one for machine learning. Spark provides data engineers and data scientists with a powerful, unified engine that is both fast and easy to use.

4. What are the languages supported by Apache Spark and which is the most popular one?

Apache Spark supports the following four languages: Scala, Java, Python and R. Among these languages, Scala and Python have interactive shells for Spark. The Scala shell can be accessed through **./bin/spark-shell** and the Python shell through **./bin/pyspark**. Scala is the most used among them because Spark is written in Scala and it is the most popularly used for Spark.

5. What are benefits of Spark over MapReduce?

Spark has the following benefits over MapReduce:

1. Due to the availability of in-memory processing, Spark implements the processing around 10 to 100 times faster than Hadoop MapReduce whereas MapReduce makes use of persistence storage for any of the data processing tasks.
2. Unlike Hadoop, Spark provides inbuilt libraries to perform multiple tasks from the same core like batch processing, Steaming, Machine learning, Interactive SQL queries. However, Hadoop only supports batch processing.
3. Hadoop is highly disk-dependent whereas Spark promotes caching and in-memory data storage.
4. Spark is capable of performing computations multiple times on the same dataset. This is called iterative computation while there is no iterative computing implemented by Hadoop.

6. What is YARN?

Similar to Hadoop, YARN is one of the key features in Spark, providing a central and resource management platform to deliver scalable operations across the cluster. YARN is a distributed container manager, like Mesos for example, whereas Spark is a data processing tool. Spark can run on YARN, the same way Hadoop Map Reduce can run on YARN. Running Spark on YARN necessitates a binary distribution of Spark as built on YARN support.

7. Do you need to install Spark on all nodes of YARN cluster?

No, because Spark runs on top of YARN. Spark runs independently from its installation. Spark has some options to use YARN when dispatching jobs to the cluster, rather than its own built-in manager, or Mesos. Further, there are some configurations to run YARN. They include *master*, *deploy-mode*, *driver-memory*, *executor-memory*, *executor-cores*, and *queue*.

8. Is there any benefit of learning MapReduce if Spark is better than MapReduce?

Yes, MapReduce is a paradigm used by many big data tools including Spark as well. It is extremely relevant to use MapReduce when the data grows bigger and bigger. Most tools like Pig and Hive convert their queries into MapReduce phases to optimize them better.

9. Explain the concept of Resilient Distributed Dataset (RDD).

RDD stands for Resilient Distribution Datasets. An RDD is a fault-tolerant collection of operational elements that run in parallel. The partitioned data in RDD is immutable and distributed in nature. There are primarily two types of RDD:

1. Parallelized Collections: Here, the existing RDDs running parallel with one another.
2. Hadoop Datasets: They perform functions on each file record in HDFS or other storage systems.

RDDs are basically parts of data that are stored in the memory distributed across many nodes. RDDs are lazily evaluated in Spark. This lazy evaluation is what contributes to Spark's speed.

10. How do we create RDDs in Spark?

Spark provides two methods to create RDD:

1. By parallelizing a collection in your Driver program.

2. This makes use of SparkContext's 'parallelize'

```
1 method val DataArray = Array(2,4,6,8,10)
2
3 val DataRDD = sc.parallelize(DataArray)
```

3. By loading an external dataset from external storage like HDFS, HBase, shared file system.

11. What is Executor Memory in a Spark application?

Every spark application has same fixed heap size and fixed number of cores for a spark executor. The heap size is what referred to as the Spark executor memory which is controlled with the spark.executor.memory property of the ***–executor-memory*** flag. Every spark application will have one executor on each worker node. The executor memory is basically a measure on how much memory of the worker node will the application utilize.

12. Define Partitions in Apache Spark.

As the name suggests, partition is a smaller and logical division of data similar to 'split' in MapReduce. It is a logical chunk of a large distributed data set. Partitioning is the process to derive logical units of data to speed up the processing process. Spark manages data using partitions that help parallelize distributed data processing with minimal network traffic for sending data between executors. By default, Spark tries to read data into an RDD from the nodes that are close to it. Since Spark usually accesses distributed partitioned data, to optimize transformation operations it creates partitions to hold the data chunks. Everything in Spark is a partitioned RDD.

13. What operations does RDD support?

RDD (Resilient Distributed Dataset) is main logical data unit in Spark. An RDD has distributed a collection of objects. Distributed means, each RDD is divided into multiple partitions. Each of these partitions can reside in memory or stored on the disk of different machines in a cluster. RDDs are immutable (Read Only) data structure. You can't change original RDD, but you can always transform it into different RDD with all changes you want.

RDDs support two types of operations: transformations and actions.

Transformations: Transformations create new RDD from existing RDD like map, reduceByKey and filter we just saw. Transformations are executed on demand. That means they are computed lazily.

Actions: Actions return final results of RDD computations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.

14. What do you understand by Transformations in Spark?

Transformations are functions applied on RDD, resulting into another RDD. It does not execute until an action occurs. map() and filter() are examples of transformations, where the former applies the function passed to it on each element of RDD and results into another RDD. The filter() creates a new RDD by selecting elements from current RDD that pass function argument.

```
val rawData=sc.textFile("path to/movies.txt")

val moviesData=rawData.map(x=>x.split("  "))
```

As we can see here, *rawData* RDD is transformed into *moviesData* RDD. Transformations are lazily evaluated.

15. Define Actions in Spark.

An action helps in bringing back the data from RDD to the local machine. An action's execution is the result of all previously created transformations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.

reduce() is an action that implements the function passed again and again until one value if left. *take()* action takes all the values from RDD to a local node.

```
moviesData.saveAsTextFile("MoviesData.txt")
```

As we can see here, *moviesData* RDD is saved into a text file called *MoviesData.txt*.

16. Define functions of SparkCore.

Spark Core is the base engine for large-scale parallel and distributed data processing. The core is the distributed execution engine and the Java, Scala, and Python APIs offer a platform for distributed ETL application development.

SparkCore performs various important functions like memory management, monitoring jobs, fault-tolerance, job scheduling and interaction with storage systems. Further, additional libraries, built atop the core allow diverse workloads for streaming, SQL, and machine learning. It is responsible for:

1. Memory management and fault recovery
2. Scheduling, distributing and monitoring jobs on a cluster
3. Interacting with storage systems

17. What do you understand by Pair RDD?

Apache defines PairRDD functions class as

```
class PairRDDFunctions[K, V] extends Logging with HadoopMapReduceUtil with Se
```

Special operations can be performed on RDDs in Spark using key/value pairs and such RDDs are referred to as Pair RDDs. Pair RDDs allow users to access each key in parallel. They have a *reduceByKey()* method that collects data based on each key and a *join()* method that combines different RDDs together, based on the elements having the same key.

18. Name the components of Spark Ecosystem.

1. **Spark Core**: Base engine for large-scale parallel and distributed data processing
2. **Spark Streaming**: Used for processing real-time streaming data
3. **Spark SQL**: Integrates relational processing with Spark's functional programming API
4. **GraphX**: Graphs and graph-parallel computation
5. **MLlib**: Performs machine learning in Apache Spark

19. How is Streaming implemented in Spark? Explain with examples.

Spark Streaming is used for processing real-time streaming data. Thus it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams. The fundamental stream unit is DStream which is basically a series of RDDs (Resilient Distributed Datasets) to process the real-time data. The data from different sources like Flume, HDFS is streamed and finally processed to file systems, live dashboards and databases. It is similar to batch processing as the input data is divided into streams like batches.

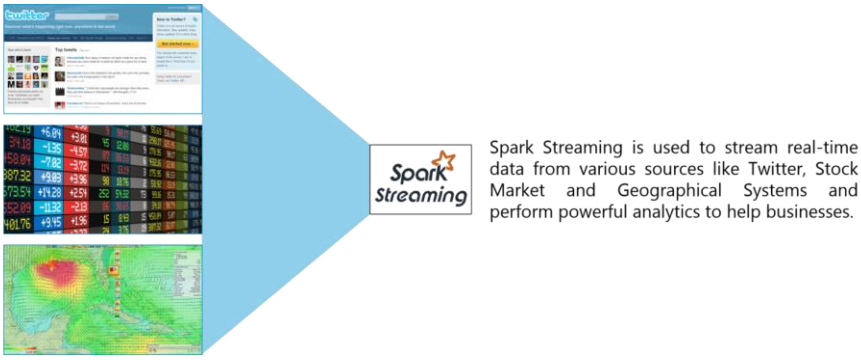


Figure: *Spark Interview Questions – Spark Streaming*

20. Is there an API for implementing graphs in Spark?

GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph.

The property graph is a directed multi-graph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. Here, the parallel edges allow multiple relationships between the same vertices. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge.

To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

21. What is PageRank in GraphX?

PageRank measures the importance of each vertex in a graph, assuming an edge from *u* to *v* represents an endorsement of *v*'s importance by *u*. For example, if a Twitter user is followed by many others, the user will be ranked highly.

GraphX comes with static and dynamic implementations of PageRank as methods on the PageRank Object. Static PageRank runs for a fixed number of iterations, while dynamic PageRank runs until the ranks converge (i.e., stop changing by more than a specified tolerance). GraphOps allows calling these algorithms directly as methods on Graph.

22. How is machine learning implemented in Spark?

MLlib is scalable machine learning library provided by Spark. It aims at making machine learning easy and scalable with common learning algorithms and use cases like clustering, regression filtering, dimensional reduction, and alike.

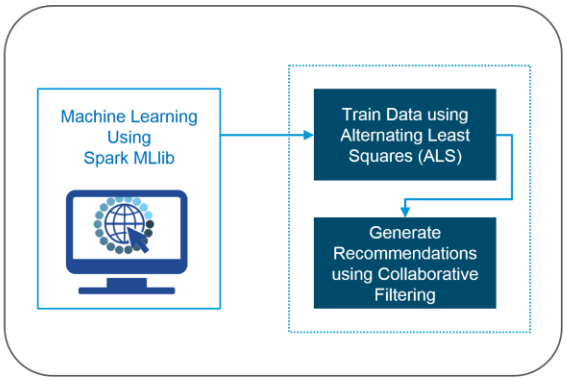


Figure: *Machine Learning Flow Diagram*

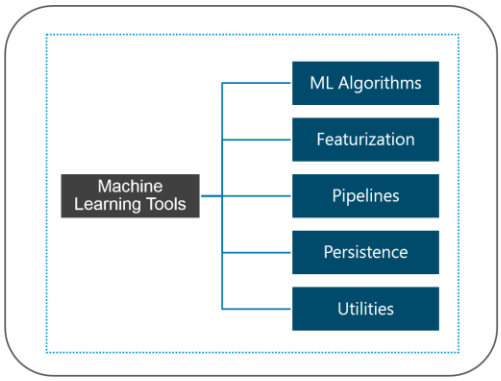


Figure: *Machine Learning Tools*

23. Is there a module to implement SQL in Spark? How does it work?

Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.

Spark SQL integrates relational processing with Spark's functional programming. Further, it provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool.

The following are the four libraries of Spark SQL.

1. Data Source API
2. DataFrame API
3. Interpreter & Optimizer
4. SQL Service

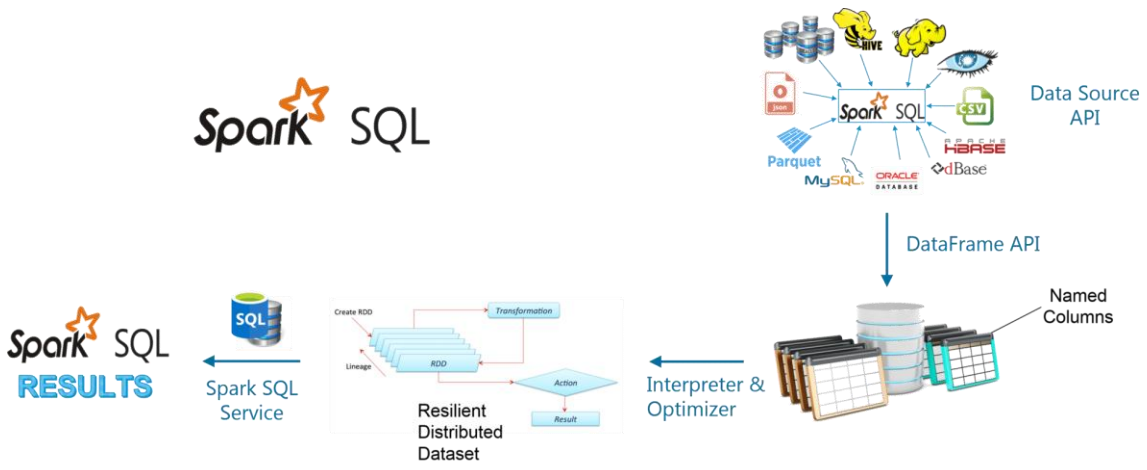


Figure: *The flow diagram represents a Spark SQL process using all the four libraries in sequence*

24. What is a Parquet file?

Parquet is a columnar format file supported by many other data processing systems. Spark SQL performs both read and write operations with Parquet file and consider it be one of the best big data analytics formats so far.

Parquet is a columnar format, supported by many data processing systems. The advantages of having a columnar storage are as follows:

1. Columnar storage limits IO operations.
2. It can fetch specific columns that you need to access.
3. Columnar storage consumes less space.
4. It gives better-summarized data and follows type-specific encoding.

25. How can Apache Spark be used alongside Hadoop?

The best part of Apache Spark is its compatibility with Hadoop. As a result, this makes for a very powerful combination of technologies. Here, we will be looking at how Spark can benefit from the best of Hadoop. Using Spark and Hadoop together helps us to leverage Spark's processing to utilize the best of Hadoop's HDFS and YARN.

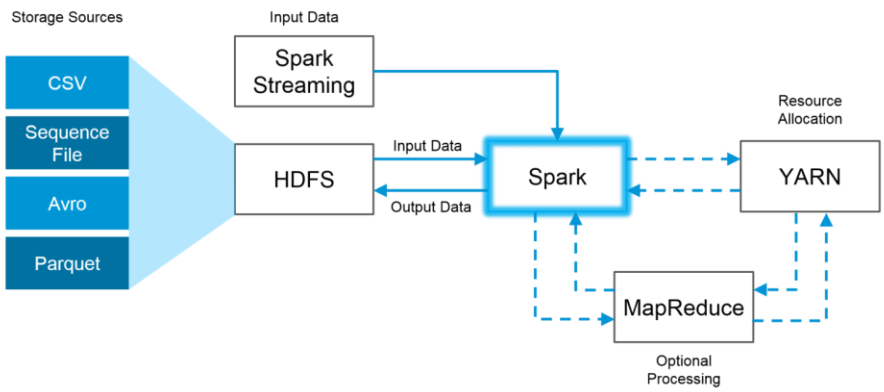


Figure: *Using Spark and Hadoop*

Hadoop components can be used alongside Spark in the following ways:

1. **HDFS**: Spark can run on top of HDFS to leverage the distributed replicated storage.
2. **MapReduce**: Spark can be used along with MapReduce in the same Hadoop cluster or separately as a processing framework.
3. **YARN**: Spark applications can also be run on YARN (Hadoop NextGen).
4. **Batch & Real Time Processing**: MapReduce and Spark are used together where MapReduce is used for batch processing and Spark for real-time processing.

26. What is RDD Lineage?

Spark does not support data replication in the memory and thus, if any data is lost, it is rebuild using RDD lineage. RDD lineage is a process that reconstructs lost data partitions. The best is that RDD always remembers how to build from other datasets.

27. What is Spark Driver?

Spark Driver is the program that runs on the master node of the machine and declares transformations and actions on data RDDs. In simple terms, a driver in Spark creates SparkContext, connected to a given Spark Master. The driver also delivers the RDD graphs to Master, where the standalone cluster manager runs.

28. What file systems does Spark support?

The following three file systems are supported by Spark:

1. Hadoop Distributed File System (HDFS).

2. Local File system.
3. Amazon S3

29. List the functions of Spark SQL.

Spark SQL is capable of:

1. Loading data from a variety of structured sources.
2. Querying data using SQL statements, both inside a Spark program and from external tools that connect to Spark SQL through standard database connectors (JDBC/ODBC). For instance, using business intelligence tools like Tableau.
3. Providing rich integration between SQL and regular Python/Java/Scala code, including the ability to join RDDs and SQL tables, expose custom functions in SQL, and more.

30. What is Spark Executor?

When SparkContext connects to a cluster manager, it acquires an Executor on nodes in the cluster. Executors are Spark processes that run computations and store the data on the worker node. The final tasks by SparkContext are transferred to executors for their execution.

31. Name types of Cluster Managers in Spark.

The Spark framework supports three major types of Cluster Managers:

1. **Standalone**: A basic manager to set up a cluster.
2. **Apache Mesos**: Generalized/commonly-used cluster manager, also runs Hadoop MapReduce and other applications.
3. **YARN**: Responsible for resource management in Hadoop.

32. What do you understand by worker node?

Worker node refers to any node that can run the application code in a cluster. The driver program must listen for and accept incoming connections from its executors and must be network addressable from the worker nodes.

Worker node is basically the slave node. Master node assigns work and worker node actually performs the assigned tasks. Worker nodes process the data stored on the node and report the resources to the master. Based on the resource availability, the master schedule tasks.

33. Illustrate some demerits of using Spark.

The following are some of the demerits of using Apache Spark:

1. Since Spark utilizes more storage space compared to Hadoop and MapReduce, there may arise certain problems.
2. Developers need to be careful while running their applications in Spark.
3. Instead of running everything on a single node, the work must be distributed over multiple clusters.
4. Spark's "in-memory" capability can become a bottleneck when it comes to cost-efficient processing of big data.
5. Spark consumes a huge amount of data when compared to Hadoop.

34. List some use cases where Spark outperforms Hadoop in processing.

1. **Sensor Data Processing**: Apache Spark's "In-memory" computing works best here, as data is retrieved and combined from different sources.
2. **Real Time Processing**: Spark is preferred over Hadoop for real-time querying of data. e.g. *Stock Market Analysis*, *Banking*, *Healthcare*, *Telecommunications*, etc.
3. **Stream Processing**: For processing logs and detecting frauds in live streams for alerts, Apache Spark is the best solution.
4. **Big Data Processing**: Spark runs upto 100 times faster than Hadoop when it comes to processing medium and large-sized datasets.

35. What is a Sparse Vector?

A sparse vector has two parallel arrays; one for indices and the other for values. These vectors are used for storing non-zero entries to save space

```
Vectors.sparse(7,Array(0,1,2,3,4,5,6),Array(1650d,50000d,800d,3.0,3.0,2009,95
```

The above sparse vector can be used instead of dense vectors.

```
val myHouse = Vectors.dense(4450d,2600000d,4000d,4.0,4.0,1978.0,95070d,1.0,1.
```

36. Can you use Spark to access and analyze data stored in Cassandra databases?

Yes, it is possible if you use Spark Cassandra Connector.To connect Spark to a Cassandra cluster, a Cassandra Connector will need to be added to the Spark project. In the setup, a Spark executor will talk to a local Cassandra node and will

only query for local data. It makes queries faster by reducing the usage of the network to send data between Spark executors (to process data) and Cassandra nodes (where data lives).

37. Is it possible to run Apache Spark on Apache Mesos?

Yes, Apache Spark can be run on the hardware clusters managed by Mesos. In a standalone cluster deployment, the cluster manager in the below diagram is a Spark master instance. When using Mesos, the Mesos master replaces the Spark master as the cluster manager. Mesos determines what machines handle what tasks. Because it takes into account other frameworks when scheduling these many short-lived tasks, multiple frameworks can coexist on the same cluster without resorting to a static partitioning of resources.

38. How can Spark be connected to Apache Mesos?

To connect Spark with Mesos:

1. Configure the spark driver program to connect to Mesos.
2. Spark binary package should be in a location accessible by Mesos.
3. Install Apache Spark in the same location as that of Apache Mesos and configure the property 'spark.mesos.executor.home' to point to the location where it is installed.

39. How can you minimize data transfers when working with Spark?

Minimizing data transfers and avoiding shuffling helps write spark programs that run in a fast and reliable manner. The various ways in which data transfers can be minimized when working with Apache Spark are:

1. Using Broadcast Variable- Broadcast variable enhances the efficiency of joins between small and large RDDs.
2. Using Accumulators – Accumulators help update the values of variables in parallel while executing.

The most common way is to avoid operations ByKey, repartition or any other operations which trigger shuffles.

40. What are broadcast variables?

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used to give every node a copy of a large input dataset in an efficient manner. Spark also

attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

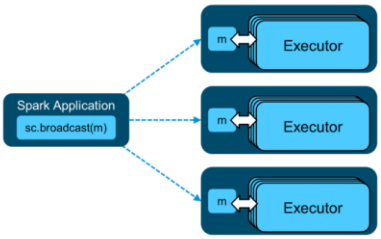


Figure: *Broadcasting A Value To Executors*

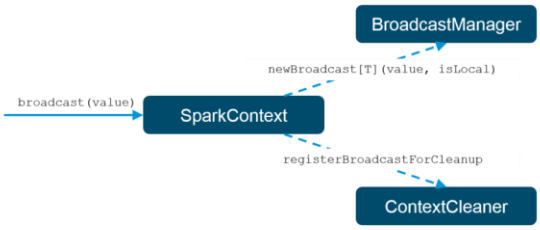
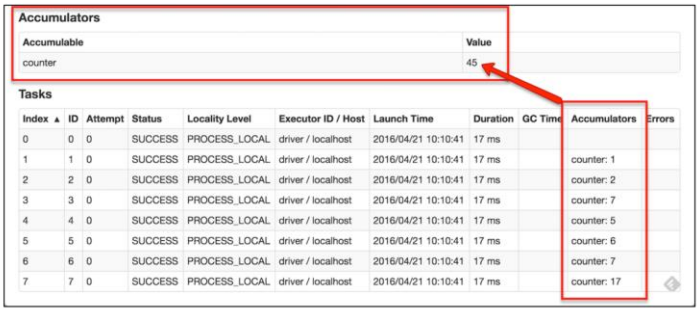


Figure: *SparkContext and Broadcasting*

41. Explain accumulators in Apache Spark.

Accumulators are variables that are only added through an associative and commutative operation. They are used to implement counters or sums. Tracking accumulators in the UI can be useful for understanding the progress of running stages. Spark natively supports numeric accumulators. We can create named or unnamed accumulators.



Accumulators

Accumulable	Value
counter	45

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Accumulators	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms			
1	1	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 1	
2	2	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 2	
3	3	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
4	4	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 5	
5	5	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 6	
6	6	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
7	7	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 17	

Figure: *Accumulators In Spark Streaming*

42. Why is there a need for broadcast variables when working with Apache Spark?

Broadcast variables are read only variables, present in-memory cache on every machine. When working with Spark, usage of broadcast variables eliminates the necessity to ship copies of a variable for every task, so data can be processed faster. Broadcast variables help in storing a lookup table inside the memory which enhances the retrieval efficiency when compared to an RDD *lookup()*.

43. How can you trigger automatic clean-ups in Spark to handle accumulated metadata?

You can trigger the clean-ups by setting the parameter '*spark.cleaner.ttl*' or by dividing the long running jobs into different batches and writing the intermediary results to the disk.

44. What is the significance of Sliding Window operation?

Sliding Window controls transmission of data packets between various computer networks. Spark Streaming library provides windowed computations where the transformations on RDDs are applied over a sliding window of data. Whenever the window slides, the RDDs that fall within the particular window are combined and operated upon to produce new RDDs of the windowed DStream.

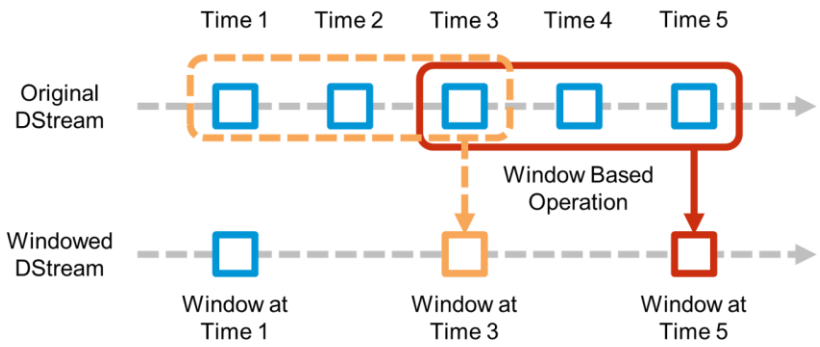


Figure: *DStream Window Transformation*

45. What is a DStream in Apache Spark?

Discretized Stream (DStream) is the basic abstraction provided by Spark Streaming. It is a continuous stream of data. It is received from a data source or from a processed data stream generated by transforming the input stream. Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval. Any operation applied on a DStream translates to operations on the underlying RDDs.

DStreams can be created from various sources like Apache Kafka, HDFS, and Apache Flume. DStreams have two operations:

1. Transformations that produce a new DStream.
2. Output operations that write data to an external system.

There are many DStream transformations possible in Spark Streaming. Let us look at **filter(*func*)**. filter(*func*) returns a new DStream by selecting only the records of the source DStream on which *func* returns true.

Figure: *Input DStream being converted through filter(func)*

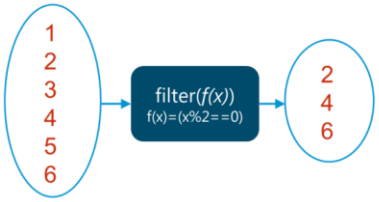


Figure: *Filter Function For Even Numbers*

46. Explain Caching in Spark Streaming.

DStreams allow developers to cache/ persist the stream's data in memory. This is useful if the data in the DStream will be computed multiple times. This can be done using the persist() method on a DStream. For input streams that receive data over the network (such as Kafka, Flume, Sockets, etc.), the default persistence level is set to replicate the data to two nodes for fault-tolerance.

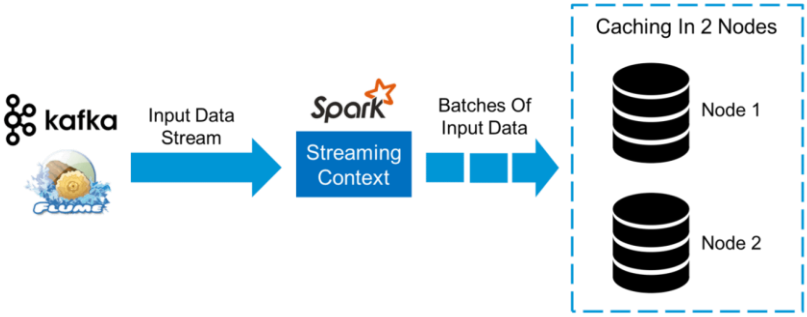


Figure: *Caching Into 2 Nodes*

47. When running Spark applications, is it necessary to install Spark on all the nodes of YARN cluster?

Spark need not be installed when running a job under YARN or Mesos because Spark can execute on top of YARN or Mesos clusters without affecting any change to the cluster.

48. What are the various data sources available in Spark SQL?

Parquet file, JSON datasets and Hive tables are the data sources available in Spark SQL.

49. What are the various levels of persistence in Apache Spark?

Apache Spark automatically persists the intermediary data from various shuffle operations, however, it is often suggested that users call persist () method on the RDD in case they plan to reuse it. Spark has various persistence levels to store the RDDs on disk or in memory or as a combination of both with different replication levels.

The various storage/persistence levels in Spark are:

1. MEMORY_ONLY: Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
2. MEMORY_AND_DISK: Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
3. MEMORY_ONLY_SER: Store RDD as *serialized* Java objects (one byte array per partition).
4. MEMORY_AND_DISK_SER: Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
5. DISK_ONLY: Store the RDD partitions only on disk.
6. OFF_HEAP: Similar to MEMORY_ONLY_SER, but store the data in off-heap memory.

50. Does Apache Spark provide checkpoints?

Checkpoints are similar to checkpoints in gaming. They make it run 24/7 and make it resilient to failures unrelated to the application logic.

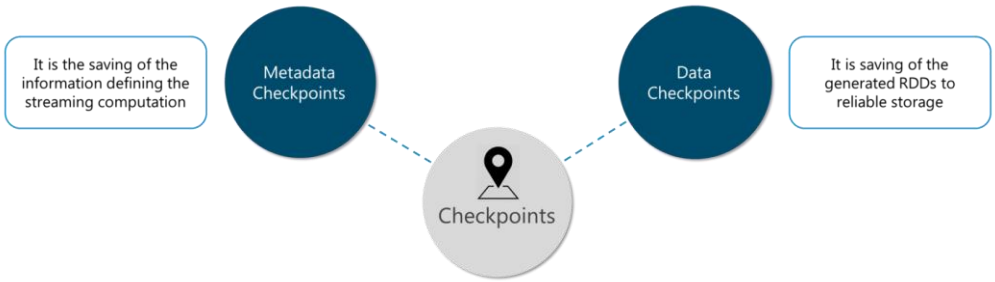


Figure: *Spark Interview Questions – Checkpoints*

Lineage graphs are always useful to recover RDDs from a failure but this is generally time-consuming if the RDDs have long lineage chains. Spark has an API for checkpointing i.e. a REPLICATE flag to persist. However, the decision on which data to checkpoint – is decided by the user. Checkpoints are useful when the lineage graphs are long and have wide dependencies.

51. How Spark uses Akka?

Spark uses Akka basically for scheduling. All the workers request for a task to master after registering. The master just assigns the task. Here Spark uses Akka for messaging between the workers and masters.

52. What do you understand by Lazy Evaluation?

Spark is intellectual in the manner in which it operates on data. When you tell Spark to operate on a given dataset, it heeds the instructions and makes a note of it, so that it does not forget – but it does nothing, unless asked for the final result. When a transformation like map*()* is called on an RDD, the operation is not performed immediately. Transformations in Spark are not evaluated till you perform an action. This helps optimize the overall data processing workflow.

Figure: *Lazy Evaluation in Apache Spark*

53. What do you understand by SchemaRDD in Apache Spark RDD?

SchemaRDD is an RDD that consists of row objects (wrappers around the basic string or integer arrays) with schema information about the type of data in each column.

SchemaRDD was designed as an attempt to make life easier for developers in their daily routines of code debugging and unit testing on SparkSQL core module. The idea can boil down to describing the data structures inside RDD using a formal description similar to the relational database schema. On top of all basic functions provided by common RDD APIs, SchemaRDD also provides some straightforward relational query interface functions that are realized through SparkSQL.

Now, it is officially renamed to *DataFrame API* on Spark's latest trunk.

54. How is Spark SQL different from HQL and SQL?

Spark SQL is a special component on the Spark Core engine that supports SQL and Hive Query Language without changing any syntax. It is possible to join SQL table and HQL table to Spark SQL.

55. Explain a scenario where you will be using Spark Streaming.

When it comes to Spark Streaming, the data is streamed in real-time onto our Spark program.

Twitter Sentiment Analysis is a real-life use case of Spark Streaming. Trending Topics can be used to create campaigns and attract a larger audience. It helps in crisis management, service adjusting and target marketing.

See Batch Details

Sentiment refers to the emotion behind a social media mention online. Sentiment Analysis is categorizing the tweets related to a particular topic and performing data mining using Sentiment Automation Analytics Tools.

Spark Streaming can be used to gather live tweets from around the world into the Spark program. This stream can be filtered using Spark SQL and then we can filter tweets based on the sentiment. The filtering logic will be implemented using MLlib where we can learn from the emotions of the public and change our filtering scale accordingly.

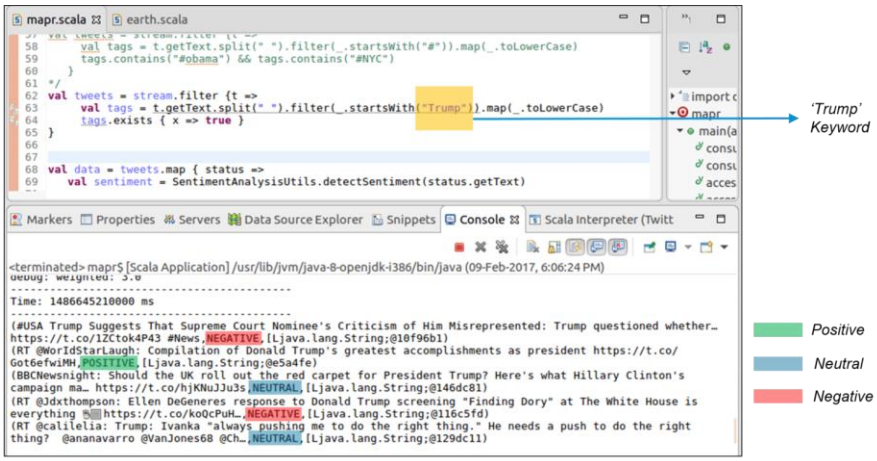


Figure: *Performing Sentiment Analysis on Tweets with 'Trump' Keyword*

The above figure displays the sentiments for the tweets containing the word *'Trump'*.