Top Answers to Spark Interview Questions

**1. Compare MapReduce and Spark?**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **MapReduce** | **Spark** |
| Processing Speeds | Good | Excellent (up to 100 times faster) |
| Data caching | Hard disk | In-memory |
| Perform iterative jobs | Average | Excellent |
| Independent of Hadoop | No | Yes |
| Machine learning applications | Average | Excellent |

**2. What is Apache Spark?**

Spark is a fast, easy-to-use and flexible data processing framework. It has an advanced execution engine supporting cyclic data  flow and in-memory computing. Spark can run on Hadoop, standalone or in the cloud and is capable of accessing diverse data sources including HDFS, HBase, Cassandra and others. Learn more in this [Apache Spark Tutorial](https://intellipaat.com/tutorial/spark-tutorial/)**.**

**3. Explain key features of Spark.**

* **Allows Integration with Hadoop and files included in HDFS.**
* **Spark has an interactive language shell as it has an independent Scala (the language in which Spark is written) interpreter.**
* **Spark consists of RDD’s (Resilient Distributed Datasets), which can be cached across computing nodes in a cluster.**
* **Spark supports multiple analytic tools that are used for interactive query analysis , real-time analysis and graph processing**

**Get Spark Certification in just 15 Hours**

**GET CERTIFIED**

**4. Define RDD?**

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

1. **Parallelized Collections :** The existing RDD’s running parallel with one another.
2. **Hadoop datasets :** perform function on each file record in HDFS or other storage system

**5. What does a Spark Engine do?**

Spark Engine is responsible for scheduling, distributing and monitoring the data application across the cluster.

**6. Define Partitions?**

As the name suggests, partition is a smaller and logical division of data  similar to ‘split’ in MapReduce. Partitioning is the process to derive logical units of data to speed up the processing process. Everything in Spark is a partitioned RDD.

[**Wish to Learn Spark? Click Here**](https://intellipaat.com/apache-spark-scala-training/#course-content)

**7. What operations RDD support?**

* Transformations.
* Actions

**8. 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 filer() 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 form current RDD that pass function argument.

**9. Define Actions.**

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. 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 local node.

**Download Spark Interview questions asked by top MNCs in 2017 ?**

Top of Form



Bottom of Form

**10. Define functions of SparkCore?**

Serving as the base engine, SparkCore performs various important functions like memory management, monitoring jobs, fault-tolerance, job scheduling and interaction with storage systems.

**11. 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.

**12. 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, 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.

***Are you interested in the comprehensive***[*Apache Spark and Scala Videos*](https://intellipaat.com/apache-spark-scala-training/#course-preview)***to take your career to the next level?***

**13. What is Hive on Spark?**

Hive contains significant support for Apache Spark, wherein Hive execution is configured to Spark:

hive> set spark.home=/location/to/sparkHome;

hive> set hive.execution.engine=spark;

Hive on Spark supports Spark on yarn mode by default.

**14. Name commonly-used Spark Ecosystems.**

* Spark SQL (Shark)- for developers.
* Spark Streaming for processing live data streams.
* GraphX for generating and computing graphs.
* MLlib (Machine Learning Algorithms).
* SparkR to promote R Programming in Spark engine.

**15. Define Spark Streaming.**

Spark supports stream processing – an extension to the Spark API , allowing stream processing of live data streams. 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.

*Learn about the Top Four Apache Spark use cases in this*[*blog post*](https://intellipaat.com/blog/top-4-apache-spark-use-cases/)*.*

**16. What is GraphX?**

Spark uses GraphX for graph processing to build and transform interactive graphs. The GraphX component enables programmers to reason about structured data at scale.

**17. What does MLlib do?**

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.

*Our in-depth*[Scala Certification](https://intellipaat.com/apache-spark-scala-training/" \l "certification" \t "_blank)*Course can give your career a big boost!*

**18. What is Spark SQL?**

SQL Spark, better known as Shark is a novel module introduced in Spark to work with structured data and perform structured data processing. Through this module, Spark executes relational SQL queries on the data. The core of the component supports an altogether different RDD called SchemaRDD, composed of rows objects and schema objects defining data type of each column in the row. It is similar to a table in relational database.

**19. 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 format so far.

**20. What file systems Spark support?**

* Hadoop Distributed File System (HDFS).  Learn more about [HDFS](https://intellipaat.com/interview-question/hdfs-interview-questions/) in these Top Interview questions.
* Local File system.
* S3

**21. 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 . Running Spark on Yarn necessitates a binary distribution of Spar as built on Yarn support.

**22. List the functions of Spark SQL.?**

Spark SQL is capable of:

* Loading data from a variety of structured sources.
* 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. Get to know more about Tableau in this [Tableau Tutorial](https://intellipaat.com/tutorial/tableau-tutorial/).
* 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.

**23. What are benefits of Spark over MapReduce?**

* Due to the availability of in-memory processing, Spark implements the processing around 10-100x faster than Hadoop MapReduce. MapReduce makes use of persistence storage for any of the data processing tasks.
* Unlike Hadoop, Spark provides in-built libraries to perform multiple tasks form the same core like batch processing, Steaming, Machine learning, Interactive SQL queries. However, Hadoop only supports batch processing.
* Hadoop is highly disk-dependent whereas Spark promotes caching and in-memory data storage.
* 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.

[Read more](https://intellipaat.com/blog/spark-vs-map-reduce/) in this blog about the comparison of Spark and MapReduce.

**24. Is there any benefit of learning MapReduce, then?**

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.

Learn more in this [MapReduce Tutorial](https://intellipaat.com/tutorial/mapreduce-tutorial/" \t "_blank).

**25. What is Spark Executor?**

When SparkContext connect 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.

**26. 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

**27. What do you understand by worker node?**

Worker node refers to any node that can run the application code in a cluster.

**28. What is PageRank?**

A unique feature and algorithm in graph, PageRank is the measure of each vertex in the graph. For instance, an edge from u to v represents endorsement of v’s importance by u. In simple terms, if a user at Instagram is followed massively, it will rank high on that platform.

**29. Do you need to install Spark on all nodes of Yarn cluster while running Spark on Yarn?**

No because Spark runs on top of Yarn.

**30. Illustrate some demerits of using Spark.**

Since Spark utilizes more storage space compared to Hadoop and MapReduce, there may arise certain problems. Developers need to be careful while running their applications in Spark. Instead of running everything on a single node, the work must be distributed over multiple clusters.

**31. How to create RDD?**

Spark provides two methods to create RDD:• By parallelizing a collection in your Driver program. This makes use of SparkContext’s ‘parallelize’ methodval IntellipaatData = Array(2,4,6,8,10)  
val distIntellipaatData = sc.parallelize(IntellipaatData)• By loading an external dataset from external storage like HDFS, [HBase](https://intellipaat.com/hbase-training/" \t "_blank), shared file system.

### ****2) What is the advantage of using Scala over other functional programming languages?****

* As the name itself indicates Scala meaning Scalable Language, its high scalable, maintainability, productivity and testability features make it advantageous to use Scala.
* Singleton and Companion Objects in Scala provide a cleaner solution unlike static in other JVM languages like Java.
* It eliminates the need for having a ternary operator as if blocks’, ‘for-yield loops’, and ‘code’ in braces return a value in Scala.

### ****3) What is the advantage of companion objects in Scala?****

Classes in Scala programming language do not have static methods or variables but rather they have what is known as a Singleton object or Companion object. The companion objects in turn are compiled to classes which have static methods.

A singleton object in Scala is declared using the keyword object as shown below –

object Main {

    def sayHello () {

        println ("Hello!");

    }

}

In the above code snippet, Main is a singleton object and the method sayHello can be invoked using the following line of code –

Main. SayHello ();

If a singleton object has the same name as that of the class then it is known as a Companion object and it should be defined in the same source file as that of the class.

class Main {

    def sayHelloWorld() {

        println("Hello World");

    }

}

object Main {

    def sayHello() {

        println("Hello!");

    }

}

#### ****Advantages of Companion Objects in Scala****

* Companion objects are beneficial for encapsulating things and they act as a bridge for writing functional and object oriented programming code.
* Using companion objects, the Scala programming code can be kept more concise as the static keyword need not be added to each and every attribute.
* Companion objects provide a clear separation between static and non-static methods in a class because everything that is located inside a companion object is not a part of the class’s runtime objects but is available from a static context and vice versa.

### ****4) Which Scala library is used for functional programming?****

Scalaz library has purely functional data structures that complement the standard Scala library. It has pre-defined set of foundational type classes like Monad, Functor, etc.

### ****5) What do you understand by “Unit” and “()” in Scala?****

Unit is a subtype of scala.anyval and is nothing but Scala equivalent of Java void that provides the Scala with an abstraction of the java platform. Empty tuple i.e. () in Scala is a term that represents unit value.

### ****6) What is the difference between concurrency and parallelism?****

People often confuse with the terms concurrency and parallelism. When several computations execute sequentially during overlapping time periods it is referred to as concurrency whereas when processes are executed simultaneously it is known as parallelism. Parallel collection, Futures and Async library are examples of achieving parallelism in Scala.

### ****7) What is a Monad in Scala?****

The simplest way to define a monad is to relate it to a wrapper. Any class object is taken wrapped with a monad in Scala. Just like you wrap any gift or present into a shiny wrapper with ribbons to make them look attractive, Monads in Scala are used to wrap objects and provide two important operations –

* Identity through “unit” in Scala
* Bind through “flatMap” in Scala

### ****8) Differentiate between Val and var in Scala.****

Val and var are the two keywords used to define variables in Scala. Var keyword is just similar to variable declaration in Java whereas Val is little different. Once a variable is declared using Val the reference cannot be changed to point to another reference. This functionality of Val keyword in Scala can be related to the functionality of java final keyword. To simplify it, Val refers to immutable declaration of a variable whereas var refers to mutable declaration of a variable in Scala.

### ****9) What do you understand by a closure in Scala?****

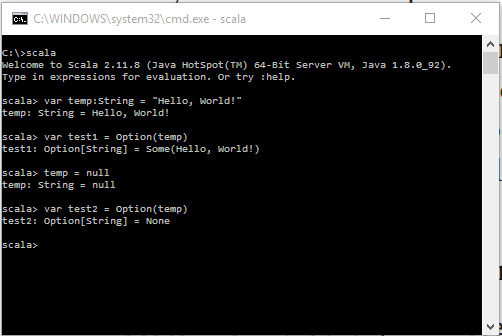
Closure is a function in Scala where the return value of the function depends on the value of one or more variables that have been declared outside the function.

### ****10) What is Scala Future? How it differs from java.util.concurrent.Future?****

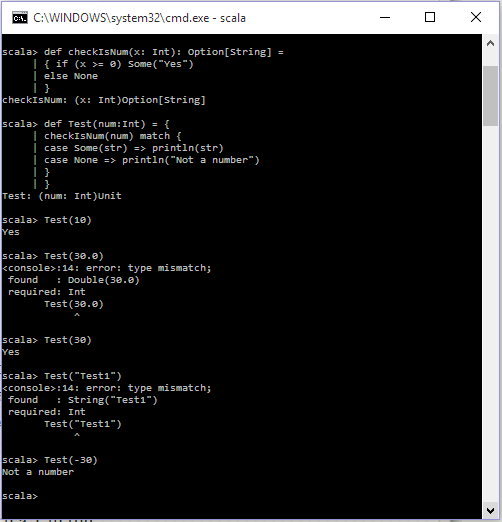
Scala Future is a monadic collection, which starts a background task. It is an object which holds the potential value or future value, which would be available after the task is completed. It also provides various operations to further chain the operations or to extract the value. Future also provide various call-back functions like onComplete, OnFailure, onSuccess to name a few, which makes Future a complete concurrent task class. The main and foremost difference between Scala’s Future and Java’s Future class is that the later does not provide promises/callbacks operations. The only way to retrieve the result is Future.get () in Java.

### ****11) What is Option in Scala? Why would you use it?****

It is used for representing whether a value is present or absent. Option collections can be used for wrapping missing values. It can also be seen as replacement for returning null values, which can be very helpful for reducing the occurrence of NullPointerException. The Option type itself is unimplemented but depends on two sub types: Some and None.



One more example to describe functionality of Option type is to use it as a method return type, which tells the caller that the method can return a string or it can return none.



### ****12) What’s the difference ‘Nil’, ‘Null’, ‘None’ and ’Nothing’ in Scala?****

* Null – It’s a sub-type of AnyRef type in Scala Types hierarchy. As Scala runs on JVM, it uses NULL to provide the compatibility with Java null keyword, or in Scala terms, to provide type for null keyword, Null type exists. It represents the absence of type information for complex types that are inherited from AnyRef.
* Nothing – It’s a sub-type of all the types exists in Scala Types hierarchy. It helps in providing the return type for the operations that can affect a normal program’s flow. It can only be used as a type, as instantiation of nothing cannot be done. It incorporates all types under AnyRef and AnyVal. Nothing is usually used as a return type for methods that have abnormal termination and result in an exception.
* Nil – It’s a handy way of initializing an empty list since, Nil, is an object, which extends List [Nothing].
* None – In programming, there are many circumstances, where we unexpectedly received null for the methods we call. In java these are handled using try/catch or left unattended causing errors in the program. Scala provides a very graceful way of handling those situations. In cases, where you don’t know, if you would be able to return a value as expected, we can use Option [T]. It is an abstract class, with just two sub-classes, Some [T] and none. With this, we can tell users that, the method might return a T of type Some [T] or it might return none.

### ****13) What is a Scala Trait?****

A trait is a special kind of Class that enables the use of multiple inheritance. Although a trait can extend only one class, but a class can have multiple traits. However, unlike classes, traits cannot be instantiated.

### ****14) When do you use Scala Traits?****

Traits are mostly used, when we require dependency injection. Unlike Java, through Spring framework, dependency injection is achieved through annotations. In Scala, there are no annotations or no special package to be imported. We just need to initialize the class with the trait and done, dependency is injected.

### ****15) What are the considerations you need to have when using Scala streams?****

Streams in Scala are a type of lazy collection, which are created using starting element and then recursively generated using those elements. Streams are like a List, except that, elements are added only when they are accessed, hence “lazy”. Since streams are lazy in terms of adding elements, they can be unbounded also, and once the elements are added, they are cached. Since Streams can be unbounded, and all the values are computed at the time of access, programmers need to be careful on using methods which are not transformers, as it may result in java.lang.OutOfMemoryErrors.

stream.max

stream.size

stream.sum

### ****16) What do you understand by diamond problem and how does Scala resolve this?****

Multiple inheritance problem is referred to as the Deadly diamond problem or diamond problem. The inability to decide on which implementation of the method to choose is referred to as the Diamond Problem in Scala. Suppose say classes B and C both inherit from class A, while class D inherits from both class B and C. Now while implementing multiple inheritance if B and C override some method from class A, there is a confusion and dilemma always on which implementation D should inherit. This is what is referred to as diamond problem. Scala resolves diamond problem through the concept of Traits and class linearization rules.

### ****17) What is tail-recursion in Scala?****

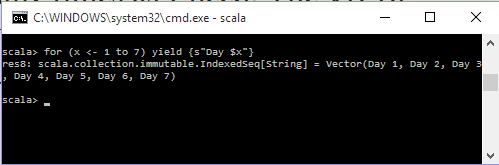
There are several situations where programmers have to write functions that are recursive in nature. The main problem with recursive functions is that, it may eat up all the allocated stack space. To overcome this situation, Scala compiler provides a mechanism “tail recursion” to optimize these recursive functions so that it does not create new stack space, instead uses the current function stack space. To qualify for this, annotation “@annotation.tailrec” has to be used before defining the function and recursive call has to be the last statement, then only the function will compile otherwise, it will give an error.

### ****18) What do you understand by Implicit Parameter?****

Wherever, we require that function could be invoked without passing all the parameters, we use implicit parameter. We provide the default values for all the parameters or parameters which we want to be used as implicit. When the function is invoked without passing the implicit parameters, local value of that parameter is used. We need to use implicit keyword to make a value, function parameter or variable as implicit.

### ****19) How does yield work in Scala?****

The yield keyword if specified before the expression, the value returned from every expression, will be returned as the collection. The yield keyword is very useful, when there is a need, you want to use the return value of expression.  The collection returned can be used the normal collection and iterate over in another loop.



### ****20) What do you understand by a case class in Scala?****

Case classes are standard classes declared with a special modifier case. Case classes export their constructor parameters and provide a recursive decomposition mechanism through pattern matching. The constructor parameters of case classes are treated as public values and can be accessed directly. For a case class, companion objects and its associated method also get generated automatically. All the methods in the class, as well, methods in the companion objects are generated based on the parameter list. The only advantage of Case class is that it automatically generates the methods from the parameter list.

#### ****Features of Case Class in Scala****

* Case objects and Case class are serializable by default.
* Case classes can be used for pattern matching.

### ****21) What is the use of Auxiliary Constructors in Scala?****

Auxiliary Constructor is the secondary constructor in Scala declared using the keywords “this” and “def”. The main purpose of using auxiliary constructors is to overload constructors. Just like in Java, we can provide implementation for different kinds of constructors so that the right one is invoked based on the requirements. Every auxiliary constructor in Scala should differ in the number of parameters or in data types.

### ****22) Differentiate between Array and List in Scala.****

* List is an immutable recursive data structure whilst array is a sequential mutable data structure.
* Lists are covariant whilst array are invariants.
* The size of a list automatically increases or decreases based on the operations that are performed on it i.e. a list in Scala is a variable-sized data structure whilst an array is fixed size data structure.

### ****23) What do you understand by apply and unapply methods in Scala?****

apply and unapply methods in Scala are used for mapping and unmapping data between form and model data.

Apply method – Used to assemble an object from its components. For example, if we want to create an Employee object  then use the two components  firstName and lastName and compose the Employee object using the apply method.

Unapply method – Used to decompose an object from its components. It follows the reverse process of apply method. So if you have an employee object, it can be decomposed into two components- firstName and lastName.

### ****24) Can a companion object in Scala access the private members of its companion class in Scala?****

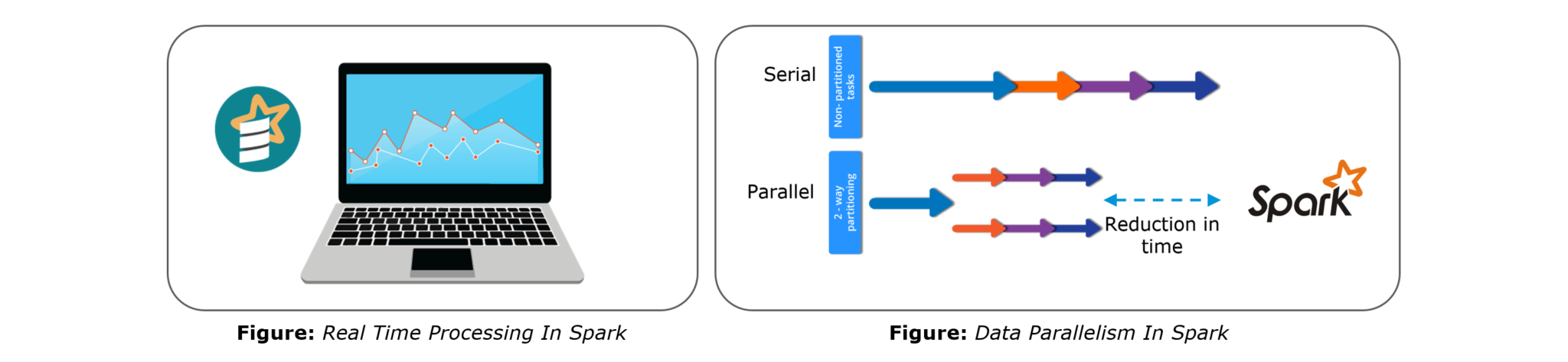
According to the private access specifier, private members can be accessed only within that class but Scala’s companion object and class provide special access to private members. A companion object can access all the private members of a companion class. Similarly, a companion class can access all the private members of companion objects.

### ****25) What is the advantage of having immutability in design for Scala programming language?****

Scala uses immutability by default in most of the cases as it helps resolve issues when dealing with concurrent programs and any other equality issues.

**1. What is Apache Spark?**

[***Apache Spark***](https://www.edureka.co/blog/spark-tutorial/?utm_source=blog&utm_medium=content-link&utm_campaign=spark-interview-questions) 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.

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!

**2. Compare MapReduce and Spark.**

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

1. **Difficulty**: Apache Spark is simpler to program and does not require any abstractions whereas MapReduce is difficult to program with abstractions.
2. **Interactivity**: Spark provides an interactive mode whereas MapReduce has no inbuilt interactive mode except for Pig and Hive.
3. **Streaming**: Spark allows real-time streaming of data and processing whereas Hadoop MapReduce can only perform batch processing on historical data.
4. **Latency**: Spark ensures lower latency computations by caching the partial results across its memory of distributed workers whereas MapReduce is completely disk-oriented.
5. **Speed**: Spark is 100 times faster than Hadoop MapReduce for big data processing as it stores the data in memory, by placing it in Resilient Distributed Databases (RDD).

*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~~*](https://www.quora.com/profile/Stan-Kladko-2)*, Galactic Exchange.io*

[**GET STARTED WITH SPARK**](https://www.edureka.co/apache-spark-scala-training?utm_source=blog&utm_medium=button&utm_campaign=spark-interview-questions)

**3. Explain key features of Spark.**

The following are the key features of Apache Spark:

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 installation 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?**

Just like in Hadoop, Yarn is one of the key features in Spark also, 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. Define 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  2 | method val DataArray = Array(2,4,6,8,10)  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.

|  |  |
| --- | --- |
| 1  2 | val rawData=sc.textFile("path to/movies.txt")  val moviesData=rawData.map(x=>x.split("\t")) |

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.

|  |  |
| --- | --- |
| 1 | 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

|  |  |
| --- | --- |
| 1 | class PairRDDFunctions[K, V] extends Logging with HadoopMapReduceUtil with Serializable |

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**: *Spark Core* is the base engine for large-scale parallel and distributed data processing.
2. **Spark Streaming**: *Spark Streaming* is used for processing real-time streaming data.
3. **Spark SQL**: *Spark SQL* is a new module in Spark which integrates relational processing with Spark’s functional programming API.
4. **GraphX**: *GraphX* is the Spark API for graphs and graph-parallel computation.
5. **MLlib**: *Spark MLlib* is used to perform 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?**

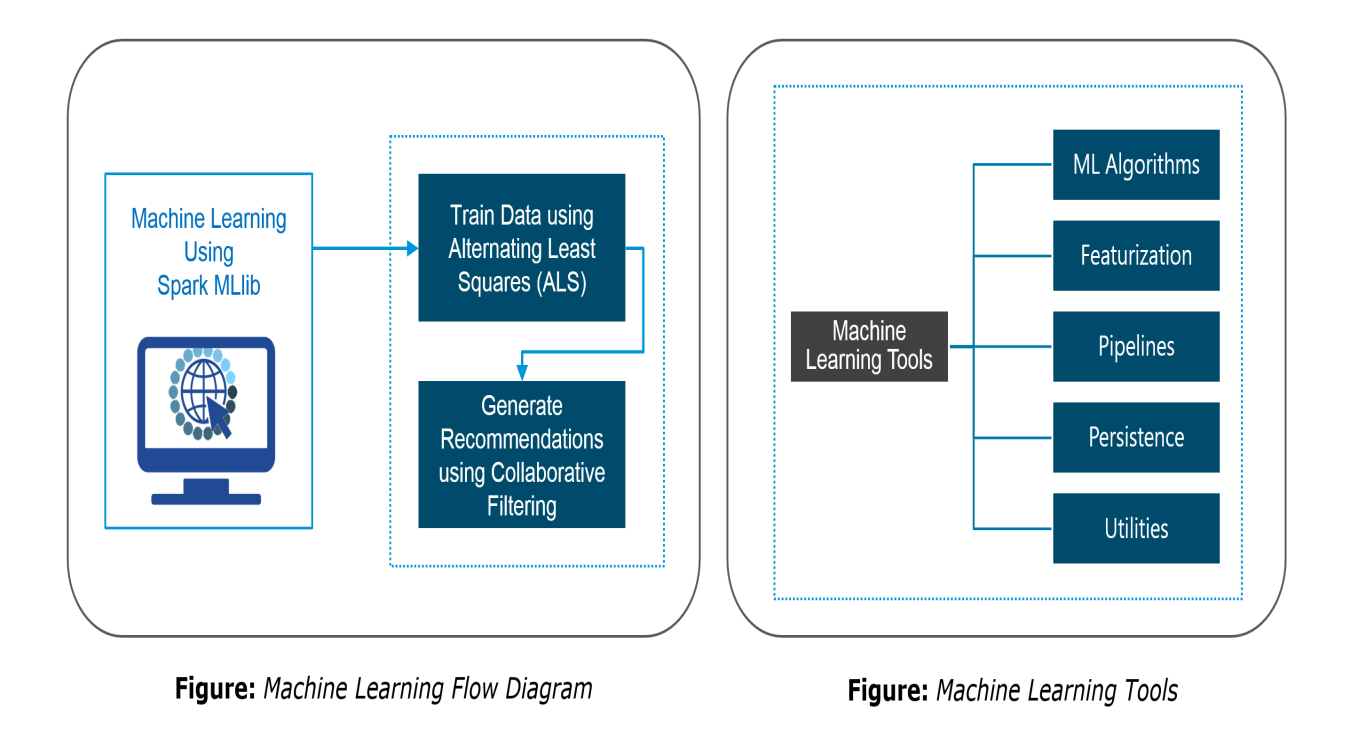
PageRank measures the importance of each vertex in a graph, assuming an edge from *u* to *v* represents an endorsement of *v*’s importance by *u*. For example, if a Twitter user is followed by many others, the user will be ranked highly.

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.

[**LEARN SPARK FROM EXPERTS**](https://www.edureka.co/apache-spark-scala-training?utm_source=blog&utm_medium=button&utm_campaign=spark-interview-questions)

**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.

****

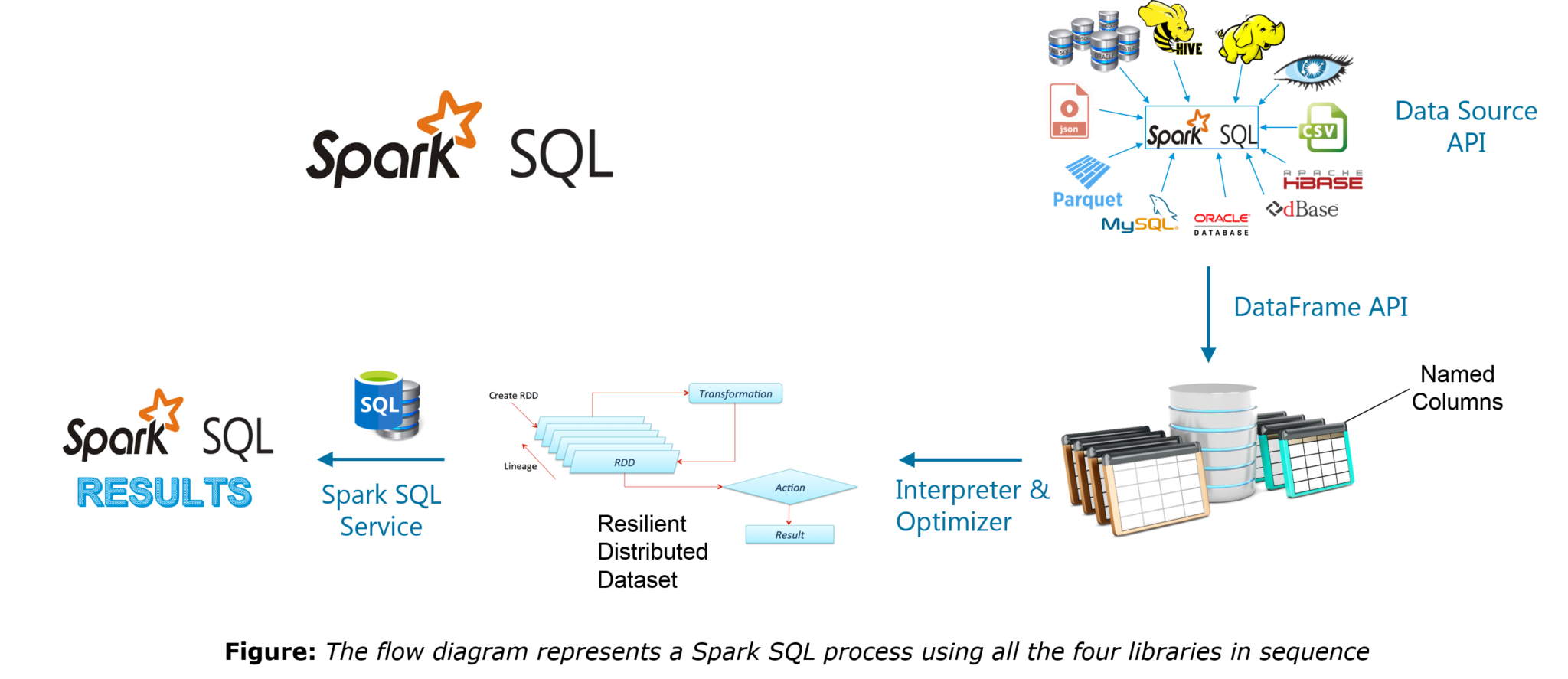
**23. Is there a module to implement SQL in Spark? How does it work?**

*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

**24. What is a Parquet file?**

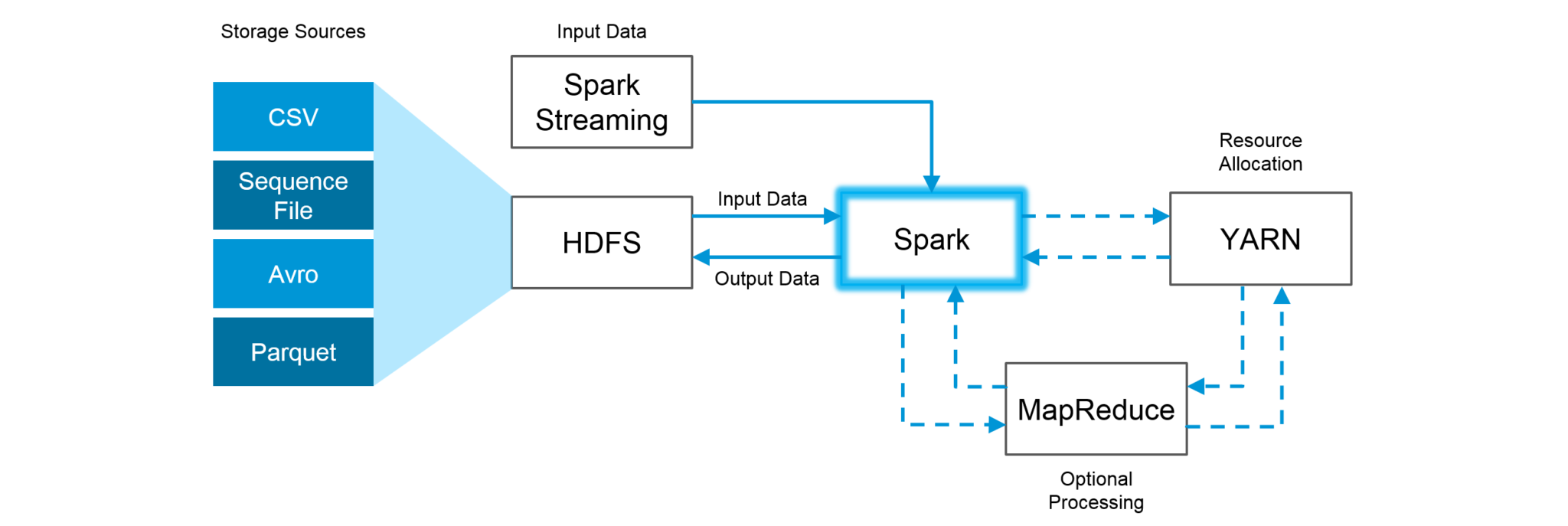
Parquet is a columnar format file supported by many other data processing systems. Spark SQL performs both read and write operations with Parquet file and consider it be one of the best big data analytics formats so far.

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 Spark be used alongside Hadoop?**

The best part of 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.

|  |  |
| --- | --- |
| 1 | Vectors.sparse(7,Array(0,1,2,3,4,5,6),Array(1650d,50000d,800d,3.0,3.0,2009,95054)) |

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

|  |  |
| --- | --- |
| 1 | val myHouse = Vectors.dense(4450d,2600000d,4000d,4.0,4.0,1978.0,95070d,1.0,1.0,1.0,0.0) |

**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?**

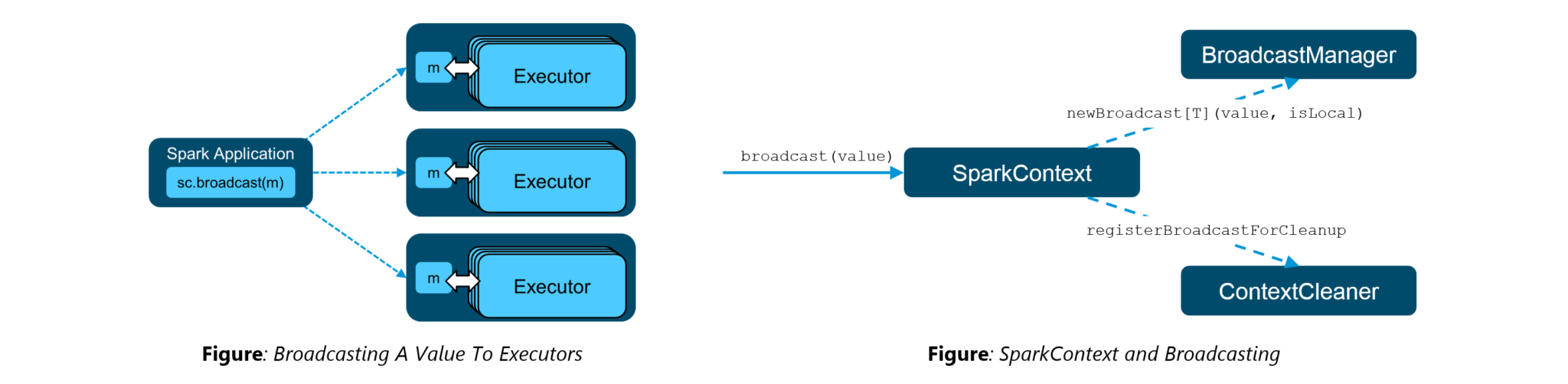
Minimizing data transfers and avoiding shuffling helps write spark programs that run in a fast and reliable manner. The various ways in which data transfers can be minimized when working with Apache Spark are:

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.

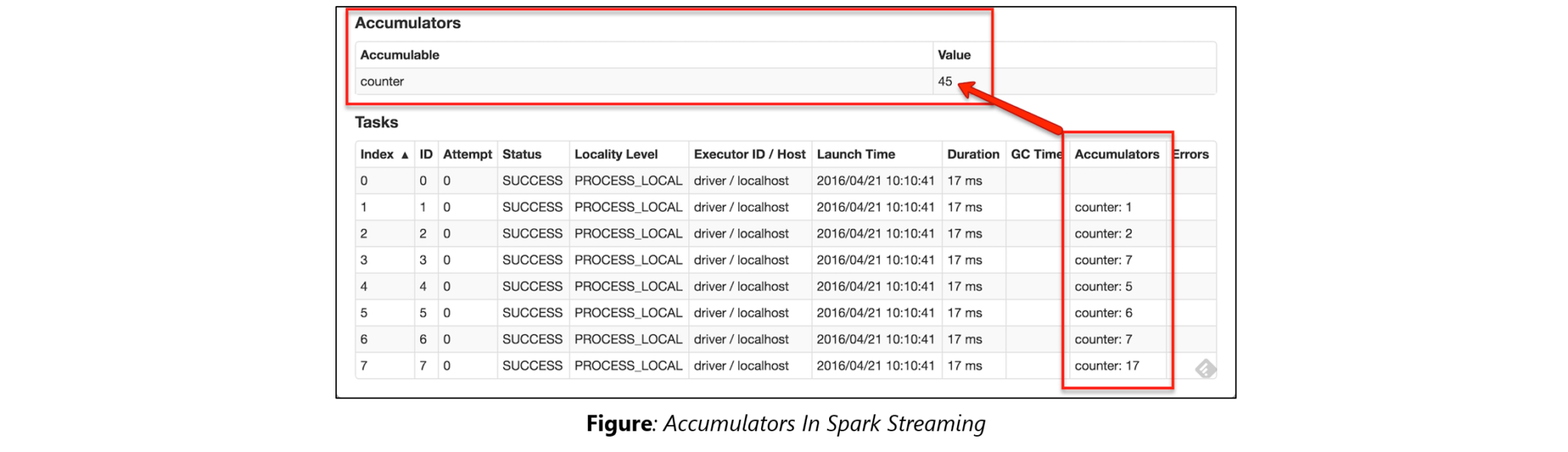
The most common way is to avoid operations ByKey, repartition or any other operations which trigger shuffles.

**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.

**41. Explain accumulators in 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.

****

**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()*.

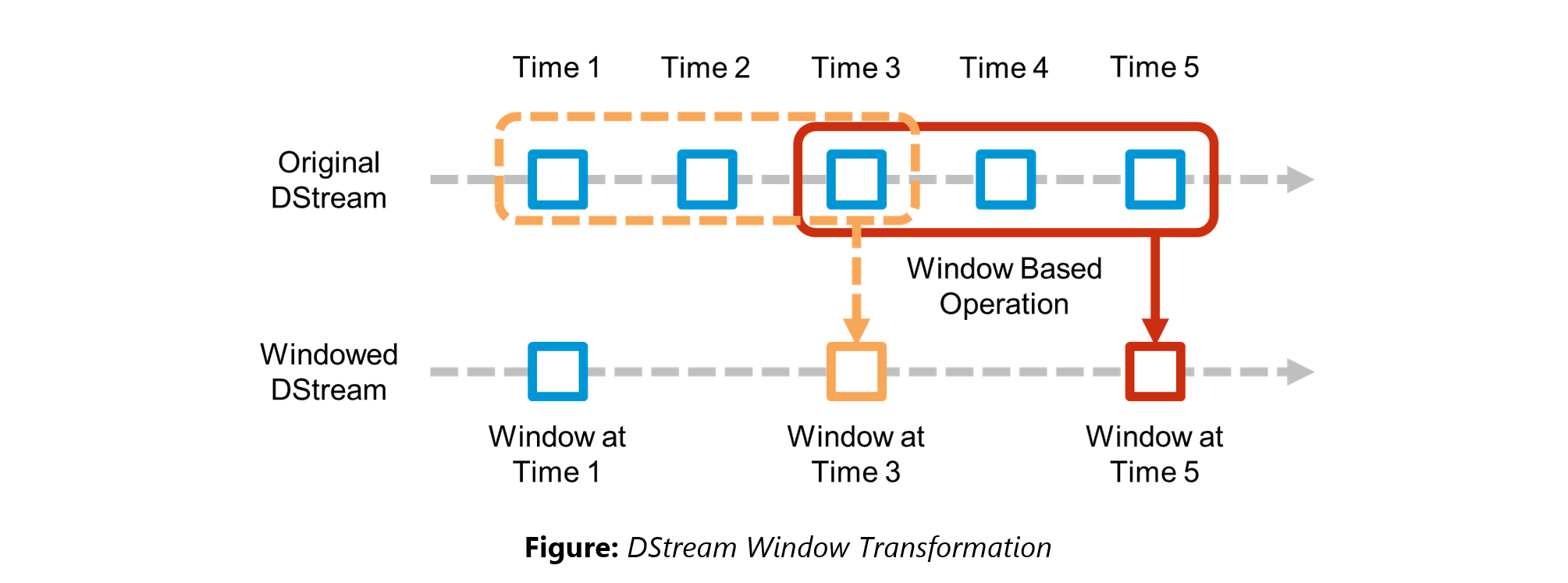
[**LEARN SPARK FROM EXPERTS**](https://www.edureka.co/apache-spark-scala-training?utm_source=blog&utm_medium=button&utm_campaign=spark-interview-questions)

**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.

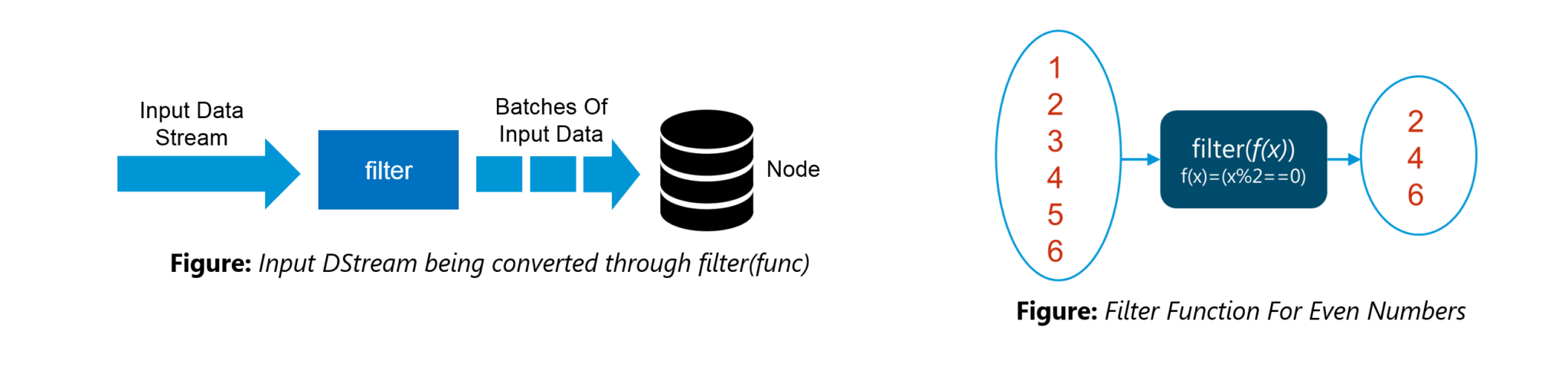
**45. What is a DStream?**

***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.

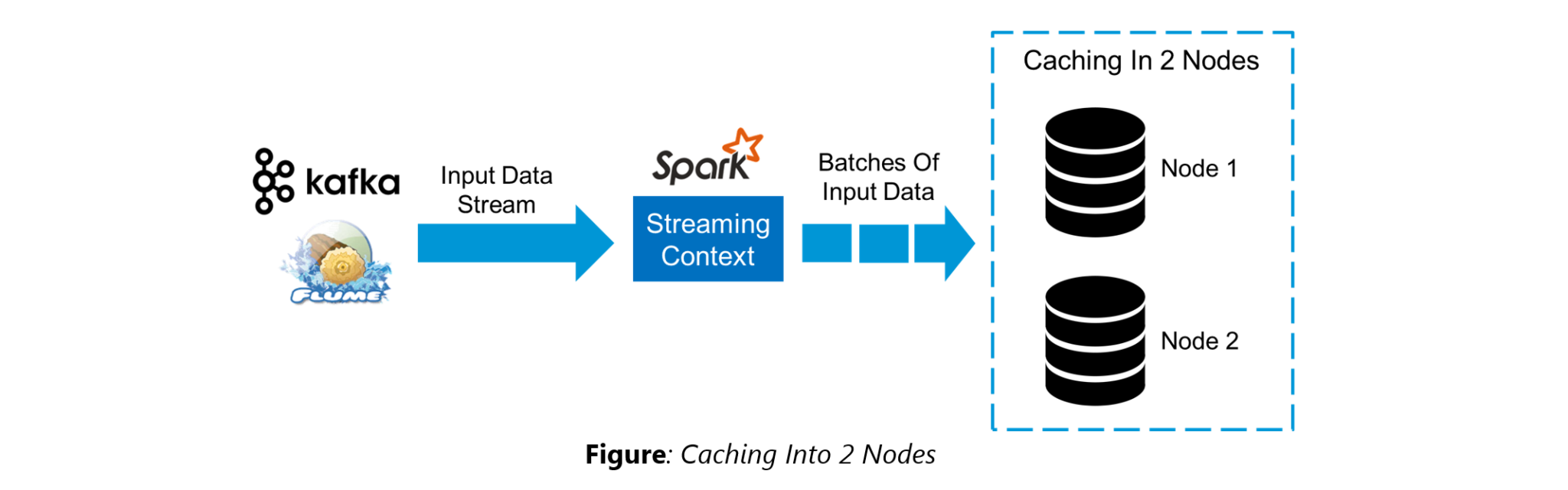
DStreams can be created from various sources like Apache Kafka, HDFS, and Apache Flume. DStreams have two operations:

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.

**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.

****

**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?**

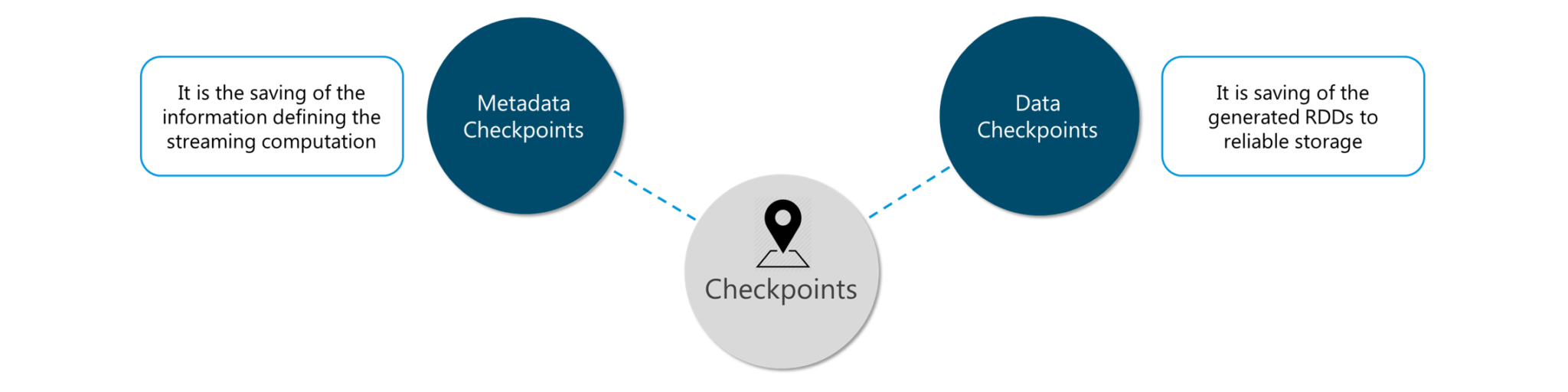
Apache Spark automatically persists the intermediary data from various shuffle operations, however, it is often suggested that users call persist () method on the RDD in case they plan to reuse it. Spark has various persistence levels to store the RDDs on disk or in memory or as a combination of both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

*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.

**53. What do you understand by SchemaRDD?**

*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.

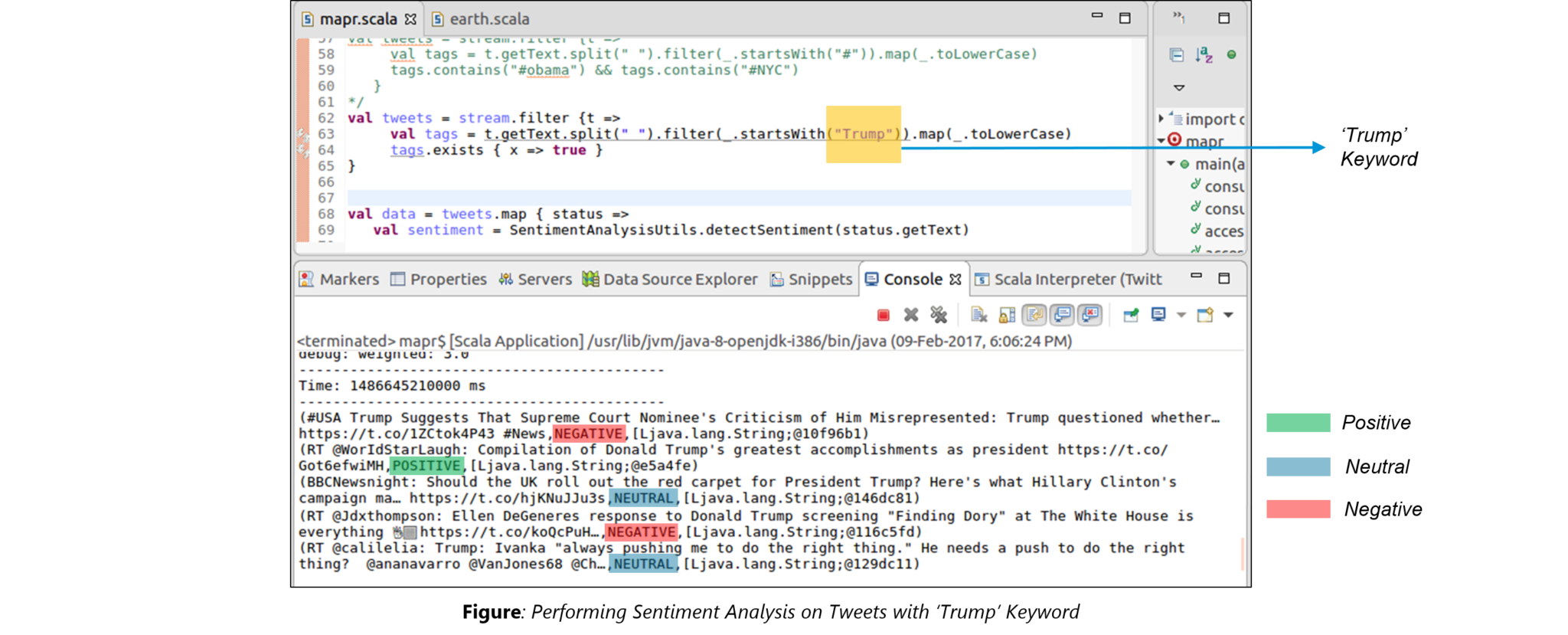
**55. Explain a scenario where you will be using Spark Streaming.**

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.

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.



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

**What is Spark?**

Spark is scheduling, monitoring and distributing engine for big data.It is a [cluster computing](http://en.wikipedia.org/wiki/Computer_cluster) platform designed to be fast and general purpose.Spark extends the popular [MapReduce](http://en.wikipedia.org/wiki/MapReduce" \o "MapReduce" \t "_blank) model.One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk.

**What is Standalone mode?**

In standalone mode, [Spark](https://tekslate.com/tutorials/spark/) uses a Master daemon which coordinates the efforts of the Workers, which run the executors. Standalone mode is the default, but it cannot be used on secure clusters.When you submit an application, you can choose how much memory its executors will use, as well as the total number of cores across all executors.

**What is YARN mode?**

In YARN mode, the YARN ResourceManager performs the functions of the Spark Master. The functions of the Workers are performed by the YARN NodeManager daemons, which run the executors. YARN mode is slightly more complex to set up, but it supports security.

**What is Apache Spark?**

Spark is a fast, easy-to-use and flexible [data processing](http://en.wikipedia.org/wiki/Computer_data_processing) framework. Most of the data users know only SQL and are not good at programming. Shark is a tool, developed for people who are from a database background – to access Scala MLib capabilities through Hive like [SQL](https://tekslate.com/sql-server-integration-in-ssis/) interface. Shark tool helps data users run Hive on Spark – offering compatibility with Hive metastore, queries and data.

**Explain key features of Spark.**

* Allows Integration with [Hadoop](https://tekslate.com/hadoop-tutorial-interview-questions-answers/" \t "_blank) and files included in HDFS.
* Spark has an interactive language shell as it has an independent Scala (the language in which Spark is written) interpreter
* Spark consists of RDD’s (Resilient Distributed Datasets), which can be cached across computing nodes in a cluster.
* Spark supports multiple analytic tools that are used for interactive query analysis , real-time analysis and graph processing.

**What are client mode and cluster mode?**

Each application has a driver process which coordinates its execution. This process can run in the foreground (**client mode**) or in the background (**cluster mode**). Client mode is a little simpler, but cluster mode allows you to easily log out after starting a Spark application without terminating the application.

**Define RDD?**

RDDs (Resilient Distributed Datasets) are basic abstraction in [Apache Spark](https://tekslate.com/tutorials/apache-scala/) that represent the data coming into the system in object format. RDDs are used for in-memory computations on large clusters, in a fault tolerant manner. RDDs are read-only portioned, collection of records, that are –  
Immutable – RDDs cannot be altered.  
Resilient – If a node holding the partition fails the other node takes the data.

**How to run spark in Standalone client mode?**

spark-submit \  
–class org.apache.spark.examples.SparkPi \  
–deploy-mode client \  
–master spark//$SPARK\_MASTER\_IP:$SPARK\_MASTER\_PORT \  
$SPARK\_HOME/examples/lib/spark-examples\_version.jar 10

**How to run spark in Standalone cluster mode?**

spark-submit \  
–class org.apache.spark.examples.SparkPi \  
–deploy-mode cluster \  
–master spark//$SPARK\_MASTER\_IP:$SPARK\_MASTER\_PORT \  
$SPARK\_HOME/examples/lib/spark-examples\_version.jar 10

**How to run spark in YARN client mode?**

spark-submit \  
–class org.apache.spark.examples.SparkPi \  
–deploy-mode client \  
–master yarn \  
$SPARK\_HOME/examples/lib/spark-examples\_version.jar 10

**How to run spark in YARN cluster mode?**

spark-submit \  
–class org.apache.spark.examples.SparkPi \  
–deploy-mode cluster \  
–master yarn \  
$SPARK\_HOME/examples/lib/spark-examples\_version.jar 10

**What is Executor memory?**

You can configure this using the –executor-memory argument to sparksubmit. Each application will have at most one executor on each worker, so this setting controls how much of that worker’s memory the application will claim. By default, this setting is 1 GB—you will likely want to increase it on most servers.

**What is the maximum number of total cores?**

This is the total number of cores used across all executors for an application. By default, this is unlimited; that is, the application will launch executors on every available node in the cluster. For a multiuser workload, you should instead ask users to cap their usage. You can set this value through the –total-execution cores argument to spark-submit, or by configuring spark.cores.max in your Spark configuration file.

**What does a Spark Engine do?**

Spark Engine is responsible for scheduling, distributing and monitoring the data application across the cluster.

**Define Partitions?**

As the name suggests, partition is a smaller and logical division of data similar to ‘split’ in [MapReduce](https://tekslate.com/tutorials/mapreduce/). Partitioning is the process to derive logical units of data to speed up the processing process. Everything in Spark is a partitioned RDD.

**What operations RDD support?**

* [Transformations](https://tekslate.com/informatica-transformations-data-ware-houses/)
* Actions

**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 filer() 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 form current RDD that pass [function argument](http://en.wikipedia.org/wiki/Parameter_%28computer_programming%29).

**Define Actions.**

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. 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 local node.

**Define functions of Spark Core?**

Spark Core performs various important functions like [memory management](https://tekslate.com/memory-management-teradata/), monitoring jobs, fault-tolerance, job scheduling and interaction with storage systems.

**What is RDD Lineage?**

Spark does not support [data replication](http://en.wikipedia.org/wiki/Replication_%28computing%29) 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.

**What is Spark**[Driver](http://en.wikipedia.org/wiki/Device_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, 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](http://en.wikipedia.org/wiki/Cluster_manager) runs.

**What is Hive on Spark?**

Hive contains significant support for Apache Spark, wherein Hive execution is configured to Spark:  
hive> set spark.home=/location/to/sparkHome;  
hive> set hive.execution.engine=spark;  
Hive on Spark supports Spark on yarn mode by default.

**Name commonly-used Spark Ecosystems?**

* Spark SQL (Shark)- for developers
* Spark Streaming for processing live data streams
* GraphX for generating and computing graphs
* MLlib (Machine Learning Algorithms)
* SparkR to promote R Programming in Spark engine.

**What are the main components of Spark?**

* **Spark Core**: Spark Core contains the basic functionality of Spark, including components for task [scheduling](https://tekslate.com/scheduling-jobs-in-datastage/), memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines RDDs,
* **Spark SQL**: Spark SQL is Spark’s package for working with structured data. It allows querying data via SQL as well as the HQL.
* **Spark Streaming:**Spark Streaming is a Spark component that enables processing of live streams of data. Examples of data streams include logfiles generated by production web servers.
* **MLlib**: Spark comes with a library containing common machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms.
* **GraphX**: GraphX is a library for manipulating graphs (e.g., a social network’s friend graph) and performing graph-parallel computations.

**How Spark Streaming works?**

Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data. DStreams can be created either from input data streams from sources such as [Kafka](https://tekslate.com/tutorials/apache-kafka-tutorials/), Flume, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

**Define Spark Streaming.Spark supports stream processing?**

An extension to the Spark API , allowing stream processing of live data streams. The data from different sources like [Flume](https://tekslate.com/tutorials/apache-flume/), HDFS is streamed and finally processed to file systems, live dashboards and databases. It is similar tobatch processing as the input data is divided into streams like batches.

**What is GraphX?**

Spark uses GraphX for graph processing to build and transform interactive graphs. The GraphX component enables programmers to reason about structured data at scale.

**What does MLlib do?**

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.

**What is Spark SQL?**

SQL Spark, better known as Shark is a novel module introduced in Spark to work with structured data and perform structured data processing. Through this module, Spark executes relational SQL queries on the data. The core of the component supports an altogether different RDD called SchemaRDD, composed of rows objects and schema objects defining data type of each column in the row. It is similar to a table in relational database.

**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 format so far.

**What file systems Spark support?**

* Hadoop Distributed File System (HDFS)
* Local File system
* S320.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 . Running Spark on Yarn necessitates a binary distribution of Spar as built on Yarn support.

**List the functions of Spark SQL?**

Spark SQL is capable of:

* Loading data from a variety of structured sources
* Querying data using SQL statements, both inside a Spark program and from external tools that connect to Spark SQL through standard database connectors ([JDBC](https://tekslate.com/jdbc-weblogic/)/ODBC). For instance, using business intelligence tools like Tableau
* 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

**What is persist()?**

Spark’s RDDs are by default recomputed each time you run an action on them. If you would like to reuse an RDD in multiple actions, you can ask Spark to persist it using RDD.persist().After computing it the first time, [Spark](https://spark.com/) will store the RDD contents in memory (partitioned across the machines in your cluster), and reuse them in future actions. Persisting RDDs on disk instead of memory is also possible.

**Write common workflow of a Spark program?**

Every Spark program and shell session will work as follows:

* Create some input RDDs from external data.
* Transform them to define new RDDs using transformations like filter().
* Ask Spark to persist() any intermediate RDDs that will need to be reused.
* Launch actions such as count() and first() to kick off a parallel computation, which is then optimized and  executed by Spark.

**Difference between cache() and persist()?**

With cache(), you use only the default storage level MEMORY\_ONLY. With persist(), you can specify which storage level you want.So cache() is the same as calling persist() with the default storage level.Spark has many levels of persistence to choose from based on what our goals are.The default persist() will store the data in the JVM heap as unserialized objects. When we write data out to disk, that data is also always serialized.Different levels of persistence are MEMORY\_ONLY, MEMORY\_ONLY\_SER, MEMORY\_AND\_DISK, MEMORY\_AND\_DISK\_SER, DISK\_ONLY.

**What is lineage graph?**

As you derive new RDDs from each other using transformations, Spark keeps track of the set of dependencies between different RDDs, called the lineage graph. It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost.

**Difference between map() and flatMap()?**

The map() transformation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD. Sometimes we want to produce multiple output elements for each input element. The operation to do this is called flatMap(). As with map(), the function we provide to flatMap() is called individually for each element in our input RDD. Instead of returning a single element, we return an iterator with our return values.

**What is reduce() action?**

It takes a function that operates on two elements of the type in your RDD and returns a new element of the same type. A simple example of such a function is +, which we can use to sum our RDD. With reduce(), we can easily sum the elements of our RDD, count the number of elements, and perform other types of aggregations.

**What is Pair RDD?**

Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDDs. Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel.For example, pair RDDs have a reduceByKey() method that can aggregate data separately for each key, and a join() method that can merge two RDDs together by grouping elements with the same key.

**What are Accumulators?**

Accumulators, provides a simple [syntax](https://tekslate.com/sql-syntax/) for aggregating values from worker nodes back to the driver program. One of the most common uses of accumulators is to count events that occur during job execution for debugging purposes.

**What is Broadcast Variables?**

Spark’s second type of shared variable, broadcast variables, allows the program to efficiently send a large, read-only value to all the worker nodes for use in one or more Spark operations. They come in handy, for example, if your application needs to send a large, read-only lookup table to all the nodes.

**What is Piping?**

Spark provides a pipe() method on RDDs. Spark’s pipe() lets us write parts of jobs using any language we want as long as it can read and write to Unix standard streams. With pipe(), you can write a transformation of an RDD that reads each RDD element from standard input as a String, manipulates that String however you like, and then writes the result(s) as Strings to standard output.

**What are benefits of Spark over MapReduce?**

* Due to the availability of in-memory processing, Spark implements the processing around 10-100x faster than  Hadoop MapReduce. MapReduce makes use of persistence storage for any of the data processing tasks.
* Unlike Hadoop, Spark provides in-built libraries to perform multiple tasks form the same core like batch  processing, Steaming, Machine learning, Interactive SQL queries. However, Hadoop only supports batch     processing.
* Hadoop is highly disk-dependent whereas Spark promotes caching and in-memory data storage
* 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.

**Is there any benefit of learning MapReduce, then?**

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.

**What is Spark Executor?**

When SparkContext connect 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.

**Name types of Cluster Managers in Spark?**

The Spark framework supports three major types of Cluster Managers:

* Standalone: a basic manager to set up a cluster
* Apache Mesos: generalized/commonly-used cluster manager, also runs Hadoop MapReduce and other applications
* Yarn: responsible for resource management in Hadoop

**What are Executors?**

Spark executors are worker processes responsible for running the individual tasks in a given Spark job. Executors are launched once at the beginning of a Spark application and typically run for the entire lifetime of an application.Executors have two roles. First, they run the tasks that make up the application and return results to the driver.Second, they provide in-memory storage for RDDs that are cached by user programs.

**What are the steps that occur when you run a Spark application on a cluster?**

The user submits an application using spark-submit.

* Spark-submit launches the driver program and invokes the main() method specified by the user.
* The driver program contacts the cluster manager to ask for resources to launch executors.
* The cluster manager launches executors on behalf of the driver program.
* The driver process runs through the user application. Based on the RDD actions and transformations in the program, the driver sends work to executors in the form of tasks.
* Tasks are run on executor processes to compute and save results.
* If the driver’s main() method exits or it calls SparkContext.stop(),it will terminate the executors and release resources from the cluster manager.

**What is Spark SQL?**

Spark SQL is a module in Apache Spark that integrates relational processing(e.g., declarative queries and optimized storage) with Spark’s procedural programming API. Spark SQL makes two main additions.First, it offers much tighter integration between relational and procedural processing, through a declarative DataFrame API.Second, it includes a highly extensible optimizer, Catalyst.

Big data applications require a mix of processing techniques, data sources and storage formats. The earliest systems designed for these workloads, such as MapReduce, gave users a powerful, but low-level, procedural programming interface. Programming such systems was onerous and required manual optimization by the user to achieve high performance. As a result, multiple new systems sought to provide a more productive user experience by offering relational interfaces to big data. Systems like Pig, Hive and Shark all take advantage of declarative queries to provide richer automatic optimizations.

**What is a schema RDD/DataFrame?**

A SchemaRDD is an RDD composed of Row objects with additional schema information of the types in each column. Row objects are just wrappers around arrays of basic types (e.g., integers and strings).

**What are Row objects?**

Row objects represent records inside SchemaRDDs, and are simply fixed-length arrays of fields.Row objects have a number of getter functions to obtain the value of each field given its index. The standard getter, get (or apply in Scala), takes a column number and returns an Object type (or Any in Scala) that we are responsible for casting to the correct type. For Boolean, Byte, Double, Float, Int, Long, Short, and String, there is a getType() method, which returns that type. For example, get String(0) would return field 0 as a string.

**What are DStreams?**

Much like Spark is built on the concept of RDDs, Spark Streaming provides an abstraction called DStreams, or discretized streams. A DStream is a sequence of data arriving over time. Internally, each DStream is represented as a sequence of RDDs arriving at each time step. DStreams can be created from various input sources, such as Flume, Kafka, or HDFS. Once built, they offer two types of operations: transformations, which yield a new DStream, and output operations, which write data to an external system.

**Explain Spark Streaming Architecture?**

Spark Streaming uses a “micro-batch” architecture, where Spark Streaming receives data from various input sources and groups it into small batches. New batches are created at regular time intervals. At the beginning of each time interval a new batch is created, and any data that arrives during that interval gets added to that batch. At the end of the time interval the batch is done growing. The size of the time intervals is determined by a parameter called the batch interval. Each input batch forms an RDD,  and is processed using Spark jobs to create other RDDs. The processed results can then be pushed out to external systems in batches.

***Desired to gain proficiency on Spark?***

***Explore the blog post on*** [*Spark training*](https://tekslate.com/spark-training) ***to become a pro in Spark.***

**What are the types of Transformations on DStreams?**

* In **stateless transformations** the processing of each batch does not depend on the data of its previous batches. They include the common RDD transformations like map(), filter(), and reduceByKey().  
  • **Stateful transformations**, in contrast, use data or intermediate results from previous batches to compute the results of the current batch. They include transformations based on sliding windows and on tracking state across time.

**What is Receiver in Spark Streaming?**

Every input DStream is associated with a **Receiver**object which receives the data from a source and stores it in Spark’s memory for processing.

**How Spark achieves fault tolerance?**

Spark stores data in-memory whereas Hadoop stores data on disk. Hadoop uses replication to achieve fault tolerance whereas Spark uses different data storage model, RDD. RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information to rebuild just that partition.This removes the need for replication to achieve fault tolerance.

**What are Spark’s main features?**

* **Speed** : Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk. Spark makes it possible by reducing number of read/write to disc. It stores this intermediate processing data in-memory. It uses the concept of an Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only it’s needed. This helps to reduce most of the disc read and write – the main time consuming factors – of data processing.
* **Combines SQL, streaming, and complex analytics:**In addition to simple “map” and “reduce” operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Not only that, users can combine all these capabilities seamlessly in a single workflow.
* **Ease of Use:**Spark lets you quickly write applications in Java, Scala, or Python. This helps developers to create and run their applications on their familiar programming languages and easy to build parallel apps.
* **Runs Everywhere:**Spark runs on Hadoop, Mesos, standalone, or in the [cloud](https://tekslate.com/cloud-foundry-training). It can access diverse data sources including HDFS, Cassandra, [HBase](https://tekslate.com/hbase-interview-questions-and-answers), S3.

**Explain about the popular use cases of Apache Spark?**

* Apache Spark is mainly used for
* Iterative machine learning.
* Interactive data analytics and processing.
* Stream processing
* Sensor data processing

**Is Apache Spark a good fit for Reinforcement learning?**

No. Apache Spark works well only for simple machine learning algorithms like clustering, regression, classification.

**What is Spark Core?**

It has all the basic functionalities of Spark, like – memory management, fault recovery, interacting with storage systems, scheduling tasks, etc.

**How can you remove the elements with a key present in any other RDD?**

Use the subtractByKey () function

**What is the difference between persist() and cache()**

persist () allows the user to specify the storage level whereas cache () uses the default storage level.

**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 –

* MEMORY\_ONLY
* MEMORY\_ONLY\_SER
* MEMORY\_AND\_DISK
* MEMORY\_AND\_DISK\_SER, DISK\_ONLY
* OFF\_HEAP

**How Spark handles monitoring and logging in Standalone mode?**

Spark has a web based user interface for monitoring the cluster in standalone mode that shows the cluster and job statistics. The log output for each job is written to the work directory of the slave nodes.

**Does Apache Spark provide checkpointing?**

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 check pointing 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.

**How can you launch Spark jobs inside Hadoop MapReduce?**

Using SIMR (Spark in MapReduce) users can run any spark job inside MapReduce without requiring any admin rights.

**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.

**How can you achieve high availability in Apache Spark?**

Implementing single node recovery with local file system Using StandBy Masters with Apache ZooKeeper.

**Hadoop uses replication to achieve fault tolerance. How is this achieved in Apache Spark?**

Data storage model in Apache Spark is based on RDDs. RDDs help achieve fault tolerance through lineage. RDD always has the information on how to build from other datasets. If any partition of a RDD is lost due to failure, lineage helps build only that particular lost partition.

**Explain about the core components of a distributed Spark application.**

* Driver- The process that runs the main () method of the program to create RDDs and perform transformations and actions on them.
* Executor –The worker processes that run the individual tasks of a Spark job.  
  Cluster Manager-A pluggable component in Spark, to launch Executors and Drivers. The cluster manager allows Spark to run on top of other external managers like Apache Mesos or YARN.

**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 a 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.

**Define a worker node?**

A node that can run the Spark application code in a cluster can be called as a worker node. A worker node can have more than one worker which is configured by setting the SPARK\_ WORKER\_INSTANCES property in the spark-env.sh file. Only one worker is started if the SPARK\_ WORKER\_INSTANCES property is not defined.

**What do you understand by SchemaRDD?**

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

We invite the big data community to share the most frequently asked Apache Spark Interview questions and answers, in the comments below – to ease big data job interviews for all prospective analytics professionals.