**What is Apache Spark? When do you use apache spark? OR What are the benefits of Spark over Mapreduce?**

Spark is really fast. If run in-memory it is 100x faster than hadoop.

In map reduce paradigm, you write many Mapreduce jobs and then tie these jobs together using Oozie/shell script. This mechanism is very time consuming and the map-reduce task have heavy latency. Between two consecutive mapreduce jobs the data has to be written to HDFS and read from HDFS. This causes a heavy latency. In case of spark, this is avoided using RDDs.

And quite often, translating the output of one MR job into the input of another MR job might require writing another code because Oozie may not suffice.

In Spark, you can basically do everything from single code or console (pyspark or scala console) and get the results immediately. Switching between 'Running something on cluster' and 'doing something locally' is fairly easy and straightforward. This also leads to less context switch of the developer and more productivity.

Spark kind of equals to MapReduce and Oozie put together.

**Is there any point of learning Mapreduce, then?**

Mapreduce is a paradigm used by many big data tools including Spark. So, understanding the MapReduce paradigm and how to convert a problem into series of MR tasks is very important.

When the data grows beyond what can fit into the memory on your cluster, the Hadoop Map-Reduce paradigm is still very relevant.

Almost, every other tool such as Hive or Pig converts its query into MapReduce phases. If you understand the Mapreduce then you will be able to optimize your queries better.

**What are the downsides of Spark?**

Spark utilizes the memory. So, in shared environment it might consume little more memory for longer durations.

The developer has to be careful. A casual developer might make following mistakes:

She may end up running everything on the local node instead of distributing work over to the cluster.

She might hit some webservice too many times by the way of using multiple clusters.

The first problem is well tackled by Hadoop Mapreduce paradigm as it ensures that the data your code is churning is fairly small a point of time thus you can make a mistake of trying to handle whole data on a single node.

The second mistake is possible in Mapreduce too. While writing Mapreduce, user may hit a service from inside of map() or reduce() too many times. This overloading of service is also possible while using Spark.

**Explain in brief what is the architecture of Spark?**

At architecture level, from a macro perspective, the Spark might look like this:

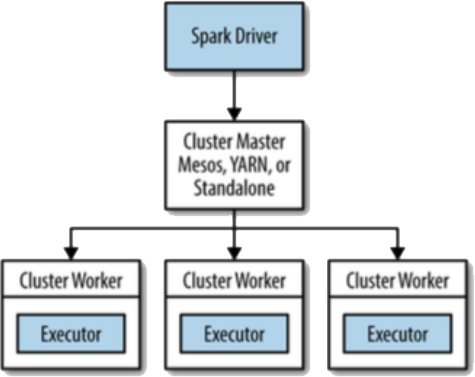
|  |
| --- |
| 5. Interactive Shells or Job Submission Layer |
| 4. API Binding: Python, Java, Scala, R, SQL |
| 3. Libraries: MLLib,, GraphX, Spark Streaming |
| 2. Spark Core (RDD & Operations on it) |
| 1. Spark Driver -> Executor |
| 0. Scheduler or Resource Manager |

**0. Scheduler or Resource Manager:**

At the bottom is the resource manager. This resource manager could be external such YARN or Mesos. Or it could be the internal if the Spark is running in standalone mode. The role of this layer is to provide a playground in which the program can run distributively. For example, YARN - Yet another resource manager would create application master, executors for any process.

**1. Spark Driver -> Executor:**

One level above scheduler is the actual code by spark which talks to the scheduler to execute. This piece of code does the real work of execution. The Spark Driver that would run inside the application master is part of this layer. Spark Driver dictates what to execute and executor executes the logic.



**2. Spark Core (RDD & Operations on it):**

Spark Core is the layer which provides maximum functionality. This layer provides abstract concepts such as RDD and the execution of the transformation and actions.

**3. Libraries: MLLib,, GraphX, Spark Streaming:**

The additional vertical wise functionalities on top of Spark Core is provided by various libraries such as MLLib, Spark Streaming, GraphX etc.

**4. API Bindings are internally calling the same API from different languages.**

**5. Interactive Shells or Job Submission Layer:**

The job submission apis provide a way to submit bundled code and interactive programs (pyspark, SparkR etc.) provide REPL or Read-Eval-Print-Loop to process data interactively.

**On which all platform can Apache Spark run?**

Spark can run on the following platforms:

YARN (Hadoop): Since yarn can handle any kind of workload, spark can run on Yarn. Though there are two modes of execution. One in which the Spark driver is executed inside the container on node and second in which the Spark driver is executed on the client machine. This is the most common way of using Spark.

Apache Mesos: Mesos is a open source good upcoming resource manager. Spark can run on mesos.

EC2: If you do not want to manage the hardware by yourself, you can run the Spark on top of Amazon EC2. This makes spark suitable for various organizations. Provides scripts that let you launch a cluster on EC2 in about 5 minutes.

Standalone:If you have no resource manager installed in your organization, you can use the standalone way. Basically, Spark provides its own resource manager. All you have to do is install Spark on all nodes in a cluster, inform each node about all nodes and start the cluster. It starts communicating with each other and run.

**What are the various modes in which Spark runs on YARN? (Local vs Client vs Cluster Mode)**

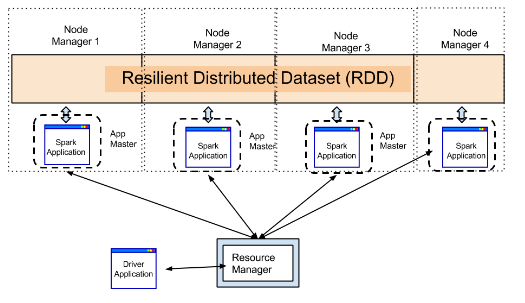
Apache Spark has two basic parts:

**1. Spark Driver**: Which controls what to execute where.

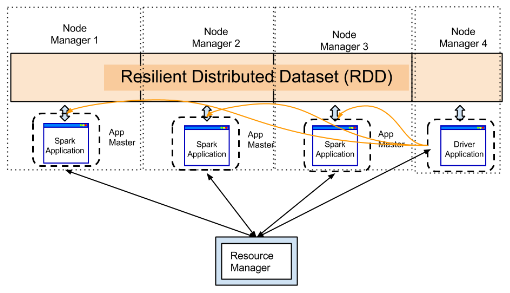
**2. Executor:** Which actually executes the logic

While running Spark on YARN, though it is very obvious that executor will be run inside containers, the driver could be run either on the machine which user is using or inside one of the containers. The first one is know is Yarn client mode while second is know is Cluster Mode. The following diagrams should give you a good idea:

**YARN client mode:** The driver runs on the machine from which client is connected



**YARN Cluster Mode:** The driver runs inside cluster.



**Local mode** is only for the case when you do not want to use a cluster and instead want to run everything on a single machine. So Driver Application and Spark Application are both on the same machine as the user.

**What are the various storages from which Spark can read data?**

Spark has been designed to process data from various sources. So, whether you want to process data stored in HDFS, Cassandra, EC2, Hive, HBASE and Tachyon. Also, it can read from any system that supports any Hadoop data source.

**While processing data from HDFS does it execute code near data?**

Yes, it does in most cases. It creates the executors near the nodes that contain data.

**Does it provide the storage layer too?**

No it doesn't provide storage layer with itself but it lets you use many data sources. Instead provides ability to read from almost every popular file systems such as HDFS, Cassandra, Hive, Hbase, SQL servers.

**Where does Spark Driver runs on Yarn?**

If you are submitting a job with --master client, the Spark driver runs on the client's machine. If you are submitting a job with --master yarn-cluster, the Spark driver would run inside a YARN container.

See question above: What are the various modes in which Spark runs on YARN? (Client vs Cluster Mode)

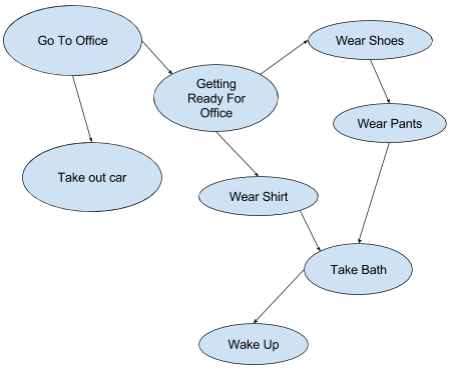
**When running Spark on Yarn, do I need to install Spark on all nodes of Yarn Cluster?**

Since spark runs as an application on top of Yarn, it utilizes yarn for the execution of its commands over the cluster's nodes. So, you do not need to install the spark on all nodes. When a job is submitted, the spark will be installed temporarily on all nodes on which execution is needed.

**What is DAG - Directed Acyclic Graph?**

Directed Acyclic Graph - DAG is a graph data structure which has edge which are directional and does not have any loops or cycles.

People use DAG almost all the time. Let's take an example of getting ready for office.



DAG is a way of representing dependencies between objects. It is widely used in computing. The examples where it is used in computing are:

1. Build tools such Apache Ant, Apache maven, make, sbt
2. Tasks Dependencies in project management - Microsoft Project
3. The data model of Git

**What is a RDD?**

The full form of RDD is resilient distributed dataset. It is a representation of data located on a network which is

Immutable - You can operate on the rdd to produce another rdd but you can’t alter it.

Partitioned / Parallel - The data located on RDD is operated in parallel. Any operation on RDD is done using multiple nodes.

Resilience - If one of the node hosting the partition fails, other nodes takes its data.

You can always think of RDD as a big array which is under the hood spread over many computers which is completely abstracted. So, RDD is made up many partitions each partition on different computers.

RDD provides two kinds of operations: Transformations and Actions.

RDD can hold data of any type of data from any supported programming languages such as Python, Java, Scala. The case where RDD's each element is a tuple - made up of (key, value)

provides extra functionalities such as "group by".

RDD is generally lazily computed i.e. it is not computed unless an action on it is called. So, RDD is either prepared out of another RDD or it is loaded from a data source. In case, it is loaded from another data source it has binding between the actual data storage and partitions. So, RDD is essentially pointer to actual data not data unless it is cached.

If a machine that hold a partition of RDD dies, the same partition is regenerated using the definition of RDD.

If there is a certain RDD that you require very frequently, you can cache it so that it is readily available instead of re-computation every time. Please note that the cached RDD will be available only during the lifetime of the application. If it is costly to recreate the RDD every time, you may want to persist it to the disc.

RDD can be stored at various data storages (such as HDFS, database etc.) in many formats

**What is lazy evaluation and how is it useful?**

Imagine there are two restaurants I (immediate) and P (patient).

In restaurant I, the waiters are very prompt - as soon as you utter the order they run to the kitchen and place order to the chef. If you have to order multiple things, the waiter will make multiple trips to the kitchen.

In P, the waiters patiently hear your all orders and once they confirm all your orders they go to the chef and place the orders. The waiter might combiner multiple dishes into one and prepare. This could lead to tremendous optimization.

While in restaurant I, the work appears to happen immediately, in restaurent P the work would be actually fast because of clubbing multiple items together. Restaurent P is doing we call 'Lazy Evaluation'.

Examples of lazy evaluations are: Spark, Pig (Pig latin). The example of immediate execution could be python interactive shell, SQL etc.

**How to create an RDD?**

You can create an RDD from an in-memory data or it from a data source such as HDFS.

You can load the data from memory in the following manner, in python:

myrdd = sc.parallelize([1,2,3,4,5]);

Here myrdd is the variable that represents an RDD created out of an in-memory object. "sc" is the spark context which is readily available if you are running in interactive mode using pyspark. Otherwise, you will have to import the SparkContext and initialize.

And to create RDD from a file in HDFS, use following:

linesrdd = sc.textFile("/data/file\_hdfs.txt");

This would create linesrdd by load a file from HDFS. Please node that this will work only if your spark is running on top of Yarn. In case, you want to load the data from external hdfs cluster, you might have to specity the protocol and namenode:

linesrdd = sc.textFile("hdfs://namenode\_host/data/file\_hdfs.txt");

In the similar fashion, you can load data from third party systems.

**When we create an RDD, does it bring the data and load it in the memory?**

No. An RDD is made up partitions which are located on multiple machines. The partition is only kept in memory if the data is being load from memory or the rdd has been cached/persisted into the memory. Otherwise, an RDD is just mapping of actual data and partitions.

**If there is certain data that we want to use again and again in different transformations, what should improve the performance?**

RDD can be persisted or cached. There are various ways in which it can be persisted: in-memory, on disc etc. So, if there is a dataset that needs a good amount computing to arrive at, you should consider caching it. You can cache it to disc if preparing it again is far more costlier than just reading from disc or it is very huge in size and would not fit in the RAM. You can cache it to memory if it can fit into the memory.

What happens to RDD when one of the nodes on which it is distributed goes down?

Since spark know how to prepare a certain dataset because it is aware of various transformations and actions that has lead to the dataset, it will be able to apply the same transformations and actions to prepare the lost partition of the node which has gone down.

**How to save RDD?**

There are few methods provided by Spark:

saveAsTextFile: Write the elements of the RDD as a text file (or set of text files) to the provided directory. The directory could be in the local filesystem, HDFS or any other file system. Each element of the dataset will be converted to text using toString() method on every element. And each element will be appended with new line character "\n"

saveAsSequenceFile:Write the elements of the dataset as a Hadoop SequenceFile. This works only on the key-value pair RDD which implement Hadoop's writable interface. You can load sequence file using sc.sequenceFile().

saveAsObjectFile: This simply saves data by serializing using standard java object serialization.

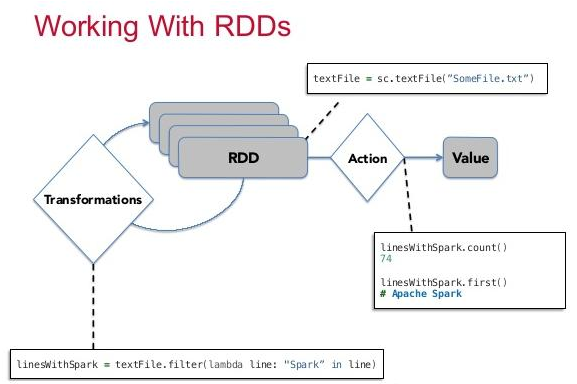
What does it mean by Columnar Storage Format?

While converting a tabular or structured data into stream of bytes we can either store rowwise or we could store column wise.

In row wise, we first store first row and then store second row and so on. In column wise, we first store first column and second column.

**When creating an RDD, what goes on internally?**

There are two ways to create RDD. One while loading data from a source. Second, by operating on existing RDD. And an action causes the computation from an RDD to yield result. The diagram below shows the relationship between RDD, transformations, actions and value/result.



**1. While loading Data from Source**

When an RDD is prepared by loading data from some source (HDFS, Cassandra, in-memory), the machines which exists nearer to the data are assigned for creation of partitions. These partitions would hold the parts of mappings or pointers to the actual data. When we are loading data from the memory (for example, by using parallelize), the partitions would hold the actual data instead of pointers or mapping.

**2. By operating on existing RDD**

An RDD is immutable. We can't change an existing RDD. We can only form a new RDD based on the previous RDD by operating on it. When operating on existing RDD, a new RDD is formed. These operations are also called transformations.

The operation could also result in shuffling - moving data across the nodes. Some operations that do not cause shuffling: map, flatmap and filter. Examples of the operations that could result in shuffling are groupByKey, repartition, sortByKey, aggregateByKey, reduceByKey, distinct.

Spark maintains the relationship between the RDD in the form of a DAG (Directed Acyclic graph). When an action such ***reduce()*** or ***saveAsTextFile()*** is called, the whole graph is evaluated and the result is returned to the driver or saved to location such as HDFS.

**What do we mean by Partitions or slices?**

Partitions (also known as slices earlier) are the parts of RDD. Each partition is generally located on a different machine. Spark runs a task for each partition during the computation. Typically, you would like to create 2-4 partitions out of your dataset per CPU in your cluster. So, if you have 4 machine with 3 CPU each, you should create the partitions between 24 (4\*3\*2) to 48 (4\*3\*4). Spark tries to create number of partitions based on your cluster configuration and the source of data.

If you are loading data from HDFS using ***textFile()***, it would create one partition per block of HDFS(64MB typically). Though you can change number of partitions by specifying second argument in the ***textFile()*** function.

If you are loading data from an existing memory using ***sc.parallelize()***, you can enforce your number of partitions by passing second argument.

You can change the number of partitions later using ***repartition()***.

If you want certain operations to consume the whole partitions at a time, you can use: ***mappartition()***.

**What does map transformation do? Provide an example.**

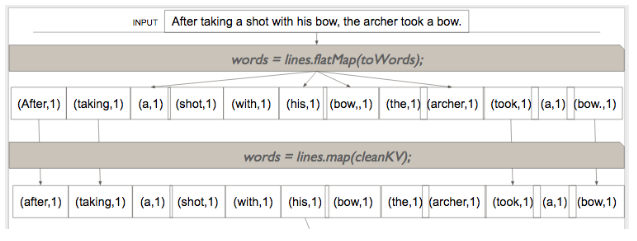
Map transformation on an RDD produces another RDD by translating each element. It translated each element by executing the function provided by the user and passing the element of the.

What is the difference between map and flatmap?

Map and flatmap both function are applied on each element of RDD. The only difference is that the function that is applied as part of map must return only one value while flatmap can return a list of values.

So, flatmap can convert one element into multiple elements of RDD while map can only result in equal number of elements.

So, if we are loading rdd from a text file, each element is a sentence. To convert this RDD into an RDD of words, we will have to apply using flatmap a function that would split a string into an array of words. If we have just to cleanup each sentence or change case of each sentence, we would be using map instead of flatmap. See the diagram below.



**What are Actions? Give some examples.**

An action brings back the data from the RDD to the local machine. Execution of an action results in all the previously created transformation. The example of actions are:

1. **reduce()** - executes the function passed again and again until only one value is left. The function should take two argument and return one value.
2. **take()** - take all the values back to the local node form RDD.

**What does reduce action do?**

A reduce action converts an RDD to a single value by applying recursively the provided (in argument) function on the elements of a RDD until only one value is left. The provided function must be commutative and associative - the order of arguments or in what way we apply the function should not make difference.

The following diagram shows the process of applying "sum" reduce function on an RDD containing 1, 2, 3, 4.

