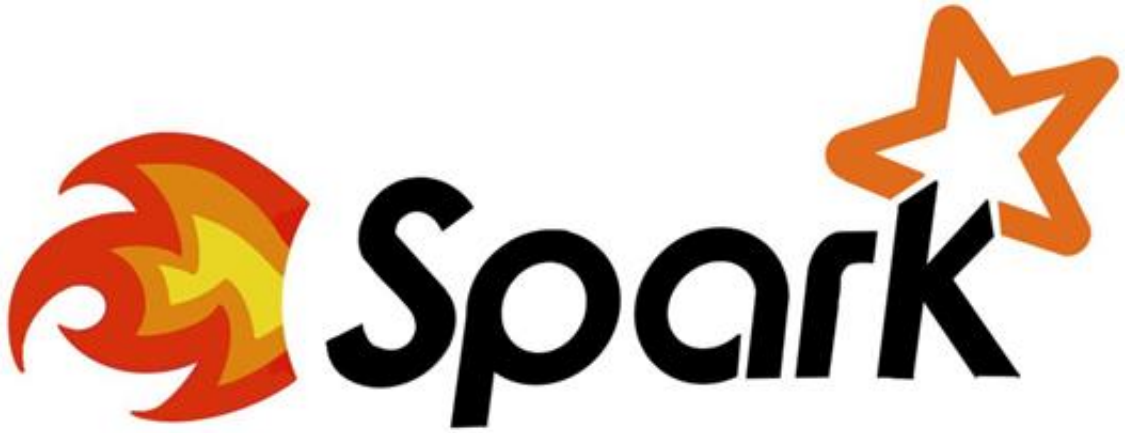
Apache Spark



WHAT IS APACHE SPARK

What is Apache Spark

- Is a fast in-memory data-processing engine
- Ability to efficiently execute streaming, machine-learning or SQL workloads which require fast-iterative access to data-sets
- Can run on top of Apache Hadoop YARN, Mesos & Kubernetes
- Is designed for data-science and its abstraction makes data-science easier
- It can cache data-set in memory and speed up iterative data-processing
- Includes ML-lib
- Is 100 folds faster than MR, in benchmark tests

Hadoop Mapreduce Limitations

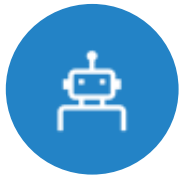
It’s based on disk based computing

Suitable for single pass computations -not iterative computations

Needs a sequence of MR jobs to run iterative tasks

Needs integration with several other frameworks/tools to solve big data use cases

Need Apache Storm for stream data processing



Need Apache Mahout for machine learning

Performance

SPARK PROCESSES DATA IN MEMORY, WHILE MR PERSISTS BACK TO DISK, AFTER A MAPREDUCE JOB

SO SPARK SHOULD OUTPERFORM MR

NONETHELESS, SPARK NEEDS A LOT OF MEMORY

IF DATA IS TOO BIG TO FIT IN MEMORY, THEN THERE WILL BE MAJOR PERFORMANCE DEGRADATION FOR SPARK

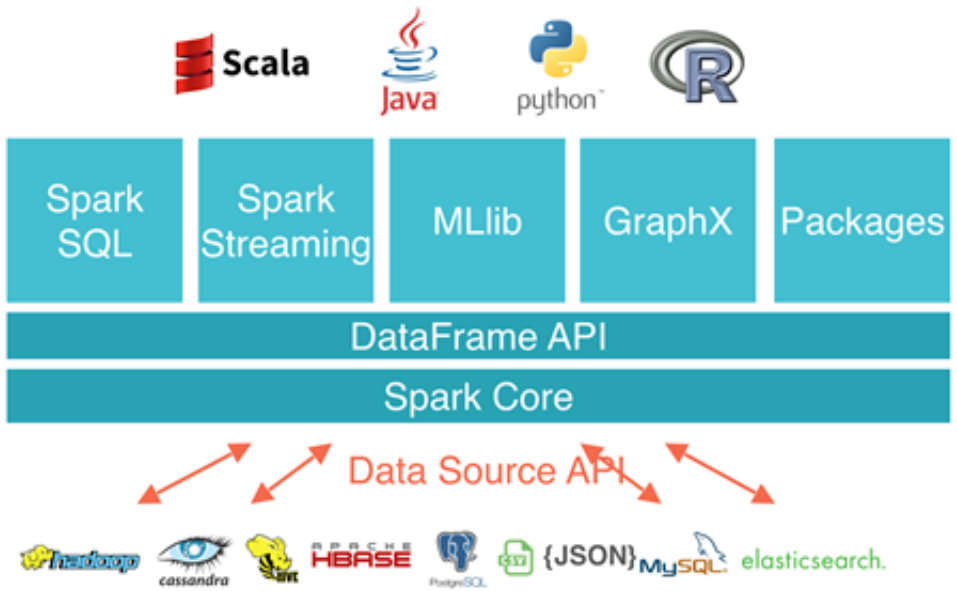
MR KILLS ITS JOB, AS SOON AS IT'S DONE

SO IT CAN RUN EASILY ALONGSIDE OTHER SERVICES WITH MINOR PERFORMANCE DIFFERENCES

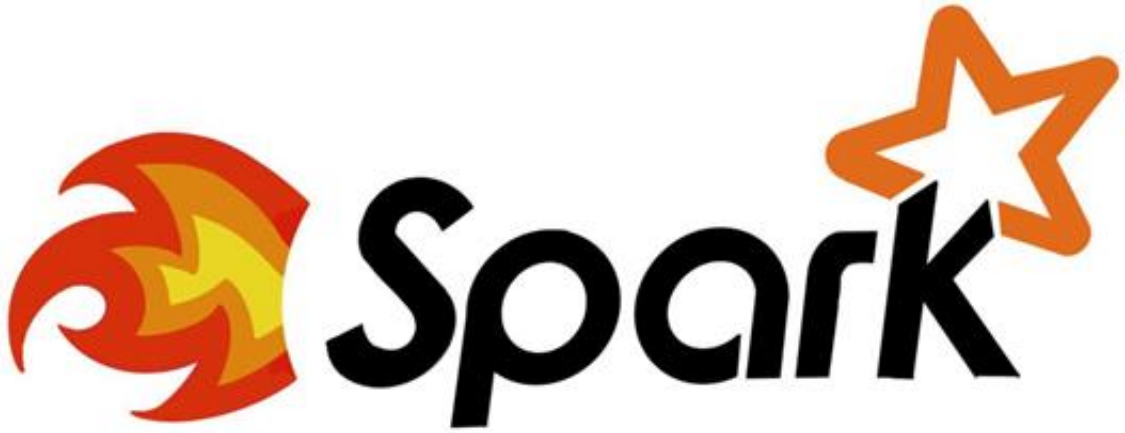
STILL SPARK HAS AN ADVANTAGE, AS LONG AS WE ARE TALKING ABOUT ITERATIVE OPERATIONS ON THE DATA

Spark Components

- **Spark Core,RDD, DF, DS, SQL –** API's for basic data processing needs for batch layer.
- **Spark Streaming-** API for real time processing needs for speed layer of data pipeline.
- **Spark MLLib –** API for Machine learning processing needs
- **Graph X-** API for needs of complex processing of Graph based data models with nodes and interactions between them.

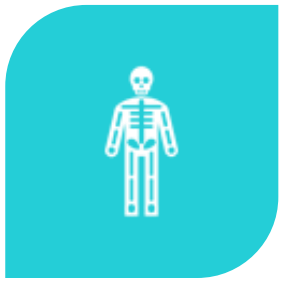


RESILIENT DISTRIBUTED DATASET (RDD)



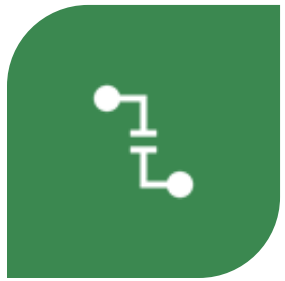
What is a RDD?

IT IS THE PRIMARY ABSTRACTION IN SPARK AND IS THE CORE OF APACHE SPARK



ONE COULD COMPARE RDDS TO COLLECTIONS IN PROGRAMMING, A RDD IS COMPUTED ON MANY JVMS WHILE A COLLECTION LIVES ON A SINGLE JVM

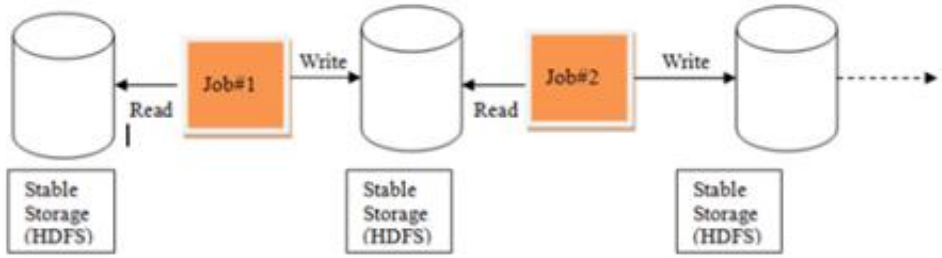
IMMUTABLE AND PARTITIONED COLLECTION OF RECORDS



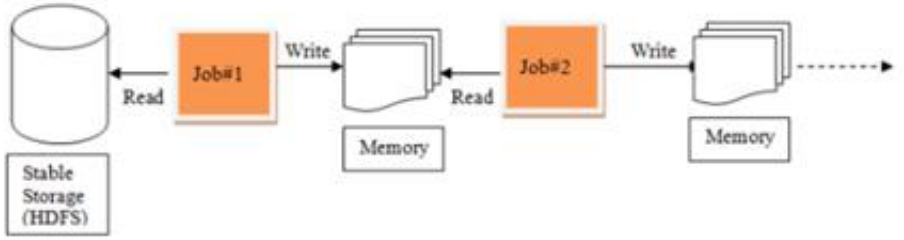
CAN ONLY BE CREATED BY READING DATA FROM A STABLE STORAGE LIKE HDFS OR BY TRANSFORMATIONS ON EXISTING RDD'S

Features of RDD

- For iterative distributed computing, it's common to reuse and share data among multiple jobs or do parallel ad-hoc queries over a shared dataset
- The persistent issue with data reuse or data sharing exists in distributed computing system(like MR) -that is, you need to store data in some intermediate stable distributed store such as HDFS or Amazon S3
- This makes overall computation of jobs slower as it involves multiple IO operations, replications and serializations in the process
- Resilient, fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures
- Distributed with data residing on multiple nodes in a cluster
- Dataset is a collection of partitioned data with primitive values or values of values, e.g tuples or other objects



Iterative processing in MR



Iterative processing in Spark

Advantages of RDD

- **How Are RDD's Fault Tolerant?**
- As RDD's are created over a set of transformations, it logs these transformations rather than actual data
- Graph of transformations to produce one RDD is called a Lineage Graph
- **For example** -firstRDD=spark.textFile("hdfs://... ")
- secondRDD=firstRDD.filter(someFunction)
- thirdRDD = secondRDD.map(someFunction)
- In case of we lose some partition of RDD , we can replay the transformation on that partition in lineage to achieve the same computation
- This is the biggest benefit of RDD , because it saves a lot of efforts in data management and replication and thus achieves faster computations

Traits of RDD

- In-Memory , data inside RDD is stored in memory as much (size) and long (time) as possible
- Immutable or Read-Only , it does not change once created and can only be transformed using transformations to new RDDs
- Lazy evaluated , the data inside RDD is not available or transformed until an action is executed that triggers the execution
- Cacheable , you can hold all the data in a persistent “storage” like memory (default and the most preferred) or disk (the least preferred due to access speed)
- Parallel, process data in parallel
- Typed-RDD records have types, Long in RDD[Long] or (Int, String) in RDD[(Int, String)]
- Partitioned-records are partitioned (split into logical partitions) and distributed across nodes in a cluster
- Location-Stickiness-RDD can define placement preferences to compute partitions (as close to the records as possible)
- RDD Supports Two Kinds of Operations
- **Actions** -operations that trigger computation and return values
- **Transformations** -lazy operations that return another RDD

Creating a RDD

Create RDDs from Python collections (lists)

```
>>>  data = [1,2,3,4,5]
>>> rDD=sc.parallelize(data,4)
>>>rDD
```

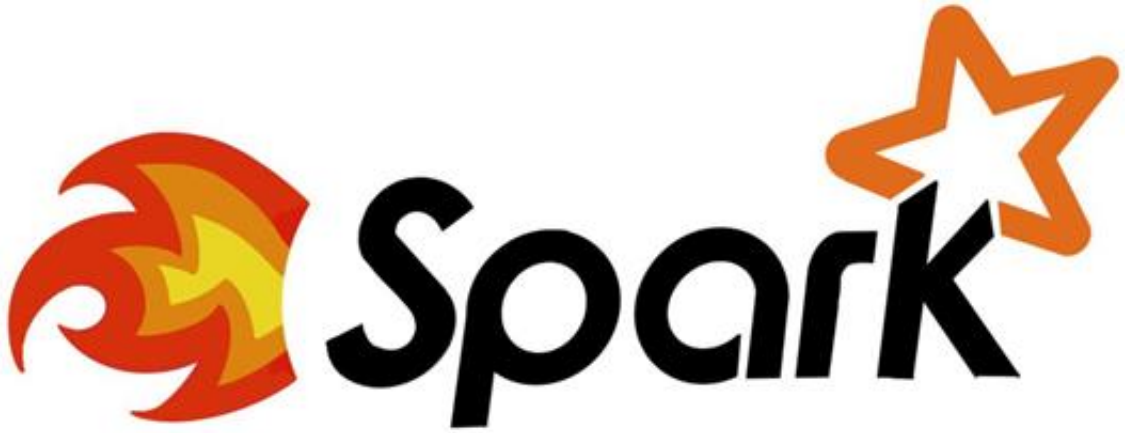
No computation occurs with sc.parallelize()

Spark only records how to create the RDD with four partitions

From HDFS, text files, Hypertable, Amazon S3, Apache Hbase,SequenceFiles, any other Hadoop InputFormat, and directory or glob wildcard: /data/201404*

```
>>> distFile = sc.textFile("README.md", 4)
>>> distFile
```

RDD Actions



ACTIONS on RDD

Actions are RDD operations that produce non-RDD values

In other words, a RDD operation that returns a value of any type except RDD[T] is an action

They trigger execution of RDD transformations to return values

Simply put, an action evaluates the RDD lineage graph

You can think of actions as a valve and until action is fired, the data to be processed is not even in the pipes, transformations

Only actions can materialize the entire processing pipeline with real data Actions are one of two ways to send data from executors to the driver (the other being accumulators)

Actions

Action	Description
`reduce(`*`func`*`)`	aggregate dataset's elements using function *func*. *func* takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
`take(`*`n`*`)`	return an array with the first *n* elements
`collect()`	return all the elements as an array WARNING: make sure will fit in driver program
`takeOrdered(`*`n, key=func`*`)`	return n elements ordered in ascending order or as specified by the optional key function

Getting Data Out of RDDs

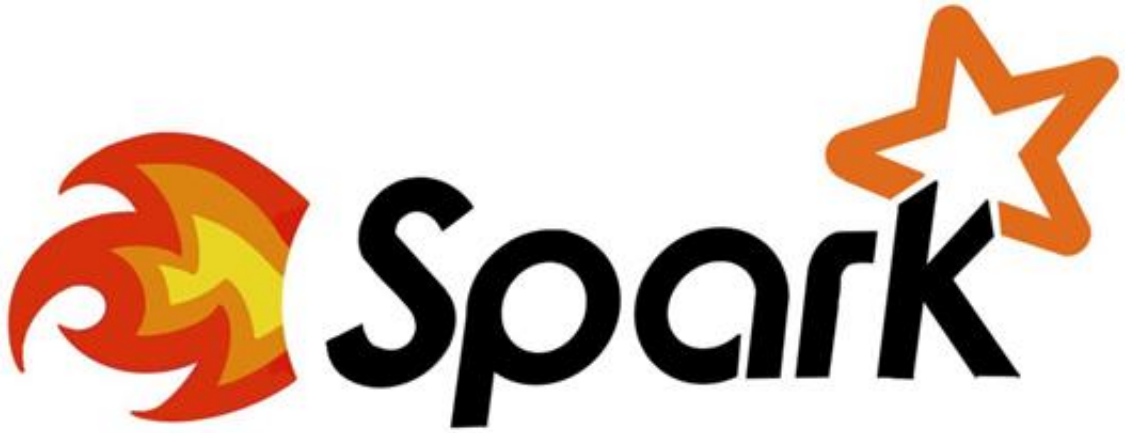
```
>>> rdd=sc.parallelize([1,2,3])
>>> rdd.reduce(lambda a,b:a*b)

>>> rdd.take(2)
>>> rdd.collect()

>>>rdd =sc.parallelize([5,3,1,2])
>>>rdd.takeOrdered(3,lambda s: -1 * s)
```

```
lines=sc.textFile("...",4)
print lines.count()
```

count() causes Spark to: read data , sum within partitions, combine sums in driver

saveAsTextFile(path) - Save this RDD as a text file, using string representations of elements.

RDD Transformations



Transformations of RDD

Transformations are lazy operations on a RDD that create one or many new RDDs, e.g. map, filter, reduceByKey, join, cogroup, randomSplit

They are functions that take a RDD as the input and produce one or many RDDs as the output They do not change the input RDD (since RDDs are immutable), but always produce one or more new RDDs by applying the computations they represent

Transformations are lazy,they are not executed immediately but only after calling an action are transformations executed

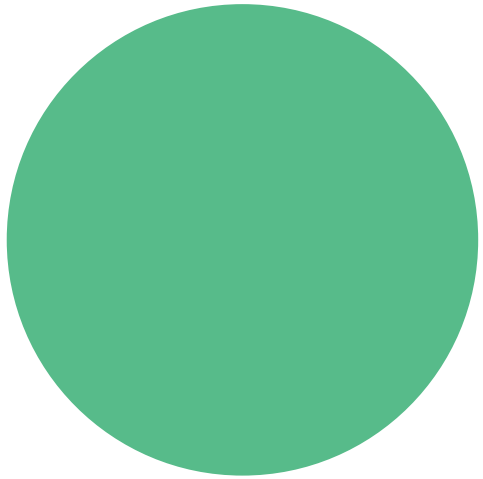
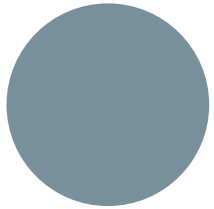
After executing a transformation, the result RDD(s) will always be different from their parents and can be smaller (e.g. filter, count, distinct, sample), bigger (e.g flatMap, union, cartesian) or the same size (e.g. map)

RDD Spark Transformations

Transformation	Description
`map(`*`func`*`)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(`*`func`*`)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([`*`numTasks`*`]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(`*`func`*`)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)

Transformations

```
>>> rdd=sc.parallelize([1,2,3,4])
>>> rdd.map(lambda x:x*2)

>>> rdd.filter(lambda x:x%2==0)

>>> rdd2=sc.parallelize([1,4,2,2,3])
>>> rdd2.distinct()
```

Thank you!