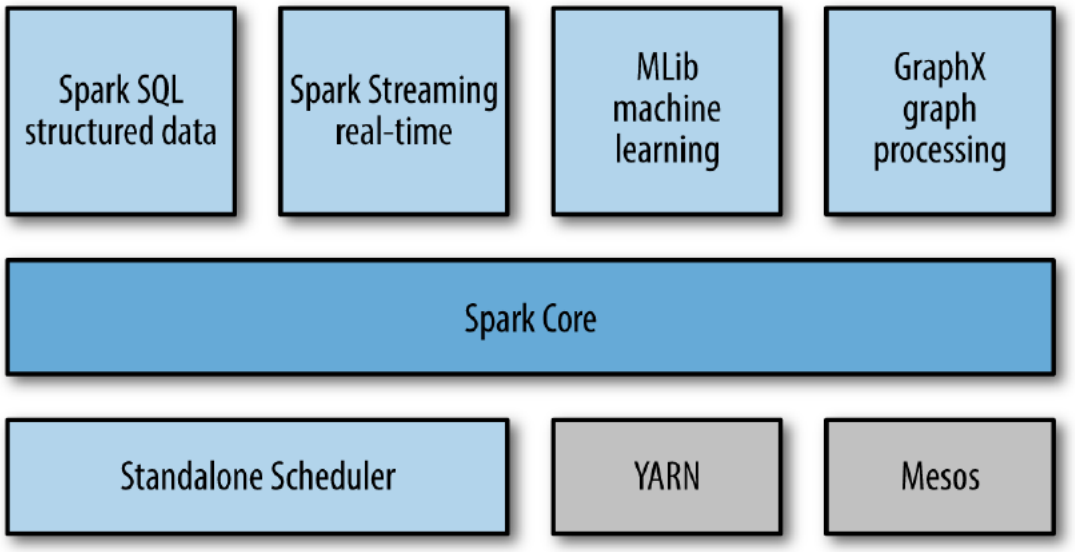
APACHE SPARK

Apache Spark is an open-source and general-purpose computing platform designed to be fast. Spark offers for speed is the ability to run computations in memory. It provides high-level APIs in Java, Python, Scala and R. It also supports a rich set of higher tools including Spark SQL for SQL and structured data processing, MLib for machine learning, GraphX for graph processing and Spark Streaming.



The Spark stack

Spark Core

It contains the basic functionality of Spark. It is responsible for scheduling, distributing, and monitoring applications. It is also the home to the API that defines RDDs (Resilient Distributed Dataset). RDDs represent a collection of items distributed across many nodes that can be manipulated in parallel.

Spark SQL

It is Spark's package for working with structured data. It allows querying of data via SQL as well as Apache Hive (variant of SQL) called the Hive Querying Language (HQL).

The input types may be Hive tables, Parquet, and JSON.

Spark Streaming

It is a Spark component that enables processing of live streams of data.

Ex: logfiles generated by production web servers or queues of messages containing status updates posted by users of a web service.

It provides an API for manipulating data streams that closely matches the Spark Core's RDD API, which makes easy for programmers to learn the project and move between applications that manipulate data stored in memory, on disk, or arriving in real time.

MLib

It comes with a library containing common machine learning (ML) functionality, called MLib. It

APACHE SPARK

provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import.

GraphX

It is a library for manipulating the graphs and performing graph parallel computations. Like Spark Streaming, Spark SQL, GraphX also extends the Spark RDD API, allowing us to create the directing graph with arbitrary properties attached to each vertex and edge. It also provides various operators for manipulating graphs (e.g., subgraph and mapVertices) and a library of common graph algorithms (e.g., PageRank and triangle counting).

Cluster Managers

Spark can run over a variety of cluster managers, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in spark itself is called the Standalone Scheduler.

Storage Layers for Spark

It can create distributed data sets from any file stored in the HDFS or other storage systems supported by Hadoop APIs (including the Local File Systems LFS, Amazon S3, Cassandra, Hive, Hbase, etc.)

Spark doesn't require hadoop; it simply has support for storage systems implementing the Hadoop APIs. The input formats are Text files, Sequence files, Avro, Parquet, and any other Hadoop Input Format.

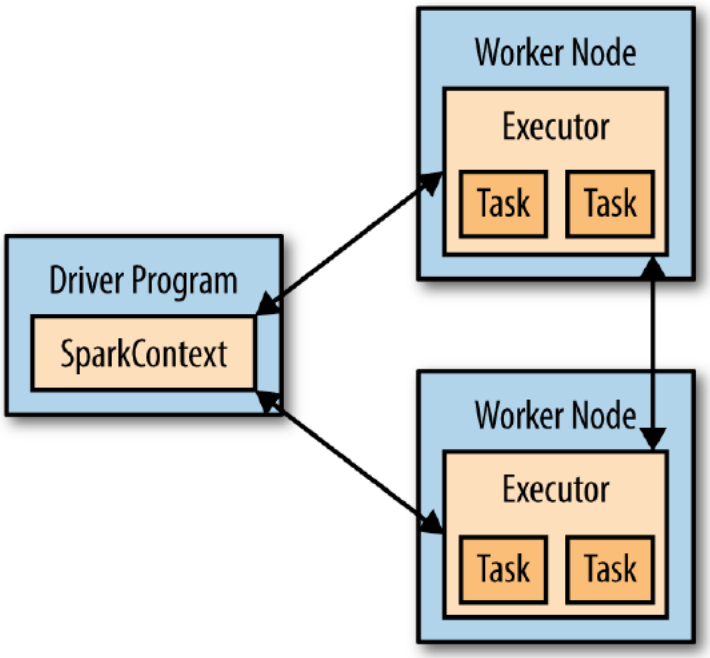
Introduction to Core Spark Concepts

Every Spark application consists of a driver program that launches various parallel operations on a cluster. The driver program contains the **main function** of the application.

Driver program access Spark through a **SparkContext** object called **sc**, which represents a connection to a computing cluster.

To run operations like collect, count, etc., driver program typically manages a number of nodes called executors.

APACHE SPARK



Components for distributed execution in spark

Programming with RDDs

An RDD is simply a distributed collection of elements. In Spark, all work is expressed as creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.

Spark automatically distributes the data contained in the RDDs across the cluster and parallelizes the operations you perform on them.

An RDD is simply an immutable distributed collection of objects.
Each RDD is split into multiple partitions, which may be computed on different nodes across the cluster.
It can contain any type of Java, Python, Scala objects, including user defined objects.

RDD can be created in 2 ways.
1) by loading an external dataset
Ex: lines = sc.textFile(“README.md”)

2) by distributing a collection of objects in your driver program
Ex: lines = sc.parallelize(["abc", "def","ghi"])

Once the RDDs are created, they offer 2 types of operations.
1) **Transformations**: Construct a new RDD from existing one
2) **Actions**: Computes the result based on the RDD, and either return it to the driver program or save it to external storage.

APACHE SPARK

At first, Actions are computed followed by Transformations. It computes in a lazy fashion.
RDDs are lazily evaluated, meaning that Spark will not begin to execute until it sees an action.

Spark's RDD are by default recomputed each time you run an action on them. If you would like to re-use an RDD in multiple actions, you can ask spark to persist it using **RDD.persist()**
cache() is the same as persist() with default storage level.
After computing it for first time, Spark will store the RDD contents in memory.
In practice, you will often use persist() to load a subset of your data into memory and query it repeatedly.

RDD Operations

1. **map()** : It returns a single element
2. **flatmap()** : Produces multiple output elements for each input element
3. **filter()** :

Pseudo set operations

4. **distinct()** :
5. **union()** :
6. **intersection()** :
7. **subtract()** :
8. **cartesian()** :

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

APACHE SPARK

RDD Actions

1. **reduce()** : takes a function that operates on 2 elements of the type in RDD and returns a new element of same type.
   (or)
   It performs concatenation i.e. we can easily sum up the elements of our RDD

2. **fold()** : It is similar to reduce() action. It takes a function with same signature as reduce(), but in addition takes a "zero value" to be used for initial call on each partition. The zero value you provide should be the identity element for your operation.
3. **aggregate()** **:** It frees us from the constraint of having the return be the same type as the RDD we are working on. We can use aggregate() to compute the average of an RDD, avoiding a map() before the fold()
4. **collect()** **:**
5. **take(n)** **:** it displays the elements from the RDD. (prints from starting of RDD)
6. **top()** **:** It displays the elements from the top of RDD. (prints from ending of RDD)
7. **count()** **:** returns the count of elements.
8. **countByValue()** **:** returns a map of each unique value to its count.

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	`4`
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`

APACHE SPARK

Function name	Purpose	Example	Result
`take(num)`	Return `num` elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top `num` elements the RDD.	`rdd.top(2)`	`{3, 3}`
`takeOrdered(num)(ordering)`	Return `num` elements based on provided ordering.	`rdd.takeOrdered(2)(myOrdering)`	`{3, 3}`
`takeSample(withReplacement, num, [seed])`	Return `num` elements at random.	`rdd.takeSample(false, 1)`	Nondeterministic
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9
`fold(zero)(func)`	Same as `reduce()` but with the provided zero value.	`rdd.fold(0)((x, y) => x + y)`	9
`aggregate(zeroValue)(seqOp, combOp)`	Similar to `reduce()` but used to return a different type.	`rdd.aggregate((0, 0))((x, y) => (x._1 + y, x._2 + 1), (x, y) => (x._1 + y._1, x._2 + y._2))`	(9, 4)
`foreach(func)`	Apply the provided function to each element of the RDD.	`rdd.foreach(func)`	Nothing

Transformations on 1 Key/Value pair RDD

1. reduceByKeyfunc() : It can aggregate the data separately for each key.
2. groupByKey() :

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey( (x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`combineBy Key(createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	
`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`
`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`
`sortByKey()`	Return an RDD sorted by the key.	`rdd.sortByKey()`	`{(1, 2), (3, 4), (3, 6)}`

Transformations on 2 pair RDDs

APACHE SPARK

Table 4-2. Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)})

Function name	Purpose	Example	Result
`subtractByKey`	Remove elements with a key present in the other RDD.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`
`cogroup`	Group data from both RDDs sharing the same key.	`rdd.cogroup(other)`	`{(1,([2],[])), (3, ([4, 6],[9]))}`

Every RDD has a fixed number of partitions that determine the degree of parallelism to use when executing the operations on the RDD.

Example 4-15. reduceByKey() with custom parallelism in Python

```
data = [("a", 3), ("b", 4), ("a", 1)]
sc.parallelize(data).reduceByKey(lambda x, y: x + y)      # Default parallelism
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10)  # Custom parallelism
```

Sometimes we want to change the partitioning of an RDD outside the context of grouping and aggregation operations. Then we use **repartition()** function, which shuffles the data across the network to create a new set of partitions. But it is expensive. So, Spark uses the optimized version of repartition() called **coalesce()** that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

To check whether you can safely call coalesce(), you can check the size of RDD using **rdd.getNumpartitions()** and make sure that you are coalescing it to fewer partitions than it currently has.

APACHE SPARK

Actions Available on Pair RDDs

Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

Data Partitioning

Spark programs can choose to control their RDDs partitioning to reduce communication. Partitioning will not be helpful in all applications – for example, if a given RDD is scanned only once, there is no point in partitioning. It is useful only when a dataset is reused multiple times in key-oriented operations such as joins.

Loading and saving your data

Spark can access the data through the InputFormat and OutputFormat interfaces used by Hadoop MapReduce, which are available for many common file formats and storage systems.

Formats
Unstructured -text
Semi structured – JSON
Structured – Sequence Files

APACHE SPARK

Table 5-1. Common supported file formats

Format name	Structured	Comments
Text files	No	Plain old text files. Records are assumed to be one per line.
JSON	Semi	Common text-based format, semistructured; most libraries require one record per line.
CSV	Yes	Very common text-based format, often used with spreadsheet applications.
SequenceFiles	Yes	A common Hadoop file format used for key/value data.
Protocol buffers	Yes	A fast, space-efficient multilanguage format.
Object files	Yes	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.

Loading text file

Input = sc.textFile("file:///home/nvemula/bin/spark/README.me")

wholeTextFiles() are useful when each file represents a certain time periods data.

Input = sc.wholeTextFiles("file:///home/nvemula/file")

The main difference, as you mentioned, is that `textFile` will return an RDD with each line as an element while `wholeTextFiles` returns a PairRDD with the key being the file path.

`wholeTextFiles` will read the complete content of a file at once, it won't be partially spilled to disk or partially garbage collected. Each file will be handled by one core and the data for each file will be one a single machine making it harder to distribute the load.

Saving text file

result.saveAsTextFile(output file path)

JSON file

Example 5-6. Loading unstructured JSON in Python

```
import json
data = input.map(lambda x: json.loads(x))
```

Example 5-9. Saving JSON in Python

```
(data.filter(lambda x: x['lovesPandas']).map(lambda x: json.dumps(x))
  .saveAsTextFile(outputFile))
```

APACHE SPARK

Table 5-2. Corresponding Hadoop Writable types

Scala type	Java type	Hadoop Writable
Int	Integer	IntWritable or VIntWritable[2]
Long	Long	LongWritable or VLongWritable[2]
Float	Float	FloatWritable
Double	Double	DoubleWritable
Boolean	Boolean	BooleanWritable
Array[Byte]	byte[]	BytesWritable
String	String	Text
Array[T]	T[]	ArrayWritable<TW>[3]
List[T]	List<T>	ArrayWritable<TW>[3]
Map[A, B]	Map<A, B>	MapWritable<AW, BW>[3]

Loading Sequence files

```
val data = sc.sequenceFile(inFile,
  "org.apache.hadoop.io.Text", "org.apache.hadoop.io.IntWritable")
```

Saving sequence files

data = sc.parallelize([“panda”, 1], [“kay”, 6], [“Snail”, 2])
data.saveAsSequenceFile(output file path and name)

File Compression

APACHE SPARK

Format	Splittable	Average compression speed	Effectiveness on text	Hadoop compression codec	Pure Java	Native	Comments
gzip	N	Fast	High	`org.apache.hadoop.io.compress.GzipCodec`	Y	Y	
lzo	Y [6]	Very fast	Medium	`com.hadoop.compression.lzo.LzoCodec`	Y	Y	LZO requires installation on every worker node
bzip2	Y	Slow	Very high	`org.apache.hadoop.io.compress.BZip2Codec`	Y	Y	Uses pure Java for splittable version
zlib	N	Slow	Medium	`org.apache.hadoop.io.compress.DefaultCodec`	Y	Y	Default compression codec for Hadoop

Format	Splittable	Average compression speed	Effectiveness on text	Hadoop compression codec	Pure Java	Native	Comments
Snappy	N	Very Fast	Low	`org.apache.hadoop.io.compress.SnappyCodec`	N	Y	There is a pure Java port of Snappy but it is not yet available in Spark/ Hadoop

Advanced Spark Programming Features

There are 2 types of shared variables
1. **Accumulators**: to aggregate information
2. **Broadcast variables**: to efficiently distribute large values.

APACHE SPARK

Numeric RDD Operations

Method	Meaning
`count()`	Number of elements in the RDD
`mean()`	Average of the elements
`sum()`	Total
`max()`	Maximum value
`min()`	Minimum value
`variance()`	Variance of the elements
`sampleVariance()`	Variance of the elements, computed for a sample
`stdev()`	Standard deviation
`sampleStdev()`	Sample standard deviation

It is a in memory based RDD application which is used for real time processes.
RDD – Resilient Distributed Dataset (In-memory, Distributed, Resilient)
It is an extension to Python list

Spark has 3

Driver – Master Node – split the job into tasks and schedule to workers
Executor – it schedules to run on workers. It is an instance which is running on workers. It is an distributed agent. It sends heart beat signal to the driver.
Worker - it keeps the overall execution of tasks. It actually stores the data which is like a data node.

Spark follows DAG structure

APACHE SPARK

RDD Operations:
1. **Transformation**: Defines a new dataset based on the previous dataset (transformations are those which are used to process the data) (These doesn't execute the logic immediately)
2. **Actions**: Which picks of a job to execute a transformation on a cluster. (These are used to preview the data) (Actions execute the logic immediately)

Lambda – It is an operator/function used to create small anonymous function.

Resilient (fault tolerance): Zero data loss and Zero down time. Spark stores data on in memory.
We can also store data on the disk only.

Spark is developed on the top of Scala.

Hadoop 1.0 only supports MapReduce based application.
Hadoop 2.0 supports non-MapReduce applications like Spark because it has YARN

Cluster management can be run on Mesos (includes non-hadoop platform also), Standalone mode and hadoop YARN

Spark can store the data on LFS, HDFS, S3, DB

Properties of RDD:

- Immutable (once created, cannot be altered) + Distributed + Catchable (Spark run on nodes which has high in memory) + Lazy evaluated (Transformation will be executed after we write Action only)
- Distributed collection of objects
- Can be reached in memory across cluster nodes
- Manipulated through various parallel operations

Term	Meaning
Application	It is a user program built on Spark. Consists of a driver program and executers on the cluster.
Application jar	A jar containing the users Spark application. In some cases, users will want to create a "world count jar" containing their application along with its dependencies. The users jar should never include Hadoop or Spark libraries; however, this will be added at runtime.
Driver program	The program running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (ex: standalone manager, Mesos, YARN)
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (ex: save, collect); you'll see this term used in drivers logs.

APACHE SPARK

Stage	Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in drivers logs.

To use python in Spark use command **pyspark**
To use Scala in Spark use command **spark-shell**

SparkContext available as **sc**, **HiveContext** available as **sqlContext**.

#created a directory named SparkSamples

[cloudera@quickstart ~]$ hadoop fs -mkdir /user/SparkSamples

#created a dataset named employeeDataSet.csv

[cloudera@quickstart ~]$ hadoop fs -touchz /user/SparkSamples/employeeDataSet.csv

#insert the data

[cloudera@quickstart ~]$ vim employeeDataSet.csv

#To view the files in the directory named SparkSamples

cloudera@quickstart ~]$ hadoop fs -ls /user/SparkSamples

Found 1 items

-rw-r--r-- 1 cloudera supergroup 0 2017-12-27 09:10 /user/SparkSamples/employeeDataSet.csv

#TO view the data in employeeDataSet.csv

[cloudera@quickstart ~]$ cat employeeDataSet.csv

Year,First Name,Country,Sex,Age

2017,NIKHIL,INDIA,M,23

2017,SHIVA,INDIA,M,23

2017,KAUSHIK,INDIA,M,28

APACHE SPARK

2017,JACK,USA,M,35

2017,JULY,USA,F,28

Transformations

Define new RDDs based on current ones

1. map() : it gives individual elements

```
>>> lines = sc.parallelize(["hadoop", "python", "spark"])
>>> wordsWithMap = lines.map(lambda line: line.split(" "))     //space should be present between quotes
>>> wordsWithMap.collect()
17/12/27 09:39:20 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 09:39:20 INFO scheduler.DAGScheduler: Got job 0 (collect at <stdin>:1) with 1 output
partitions
17/12/27 09:39:20 INFO scheduler.DAGScheduler: Final stage: ResultStage 0(collect at <stdin>:1)
17/12/27 09:39:20 INFO scheduler.DAGScheduler: Job 0 finished: collect at <stdin>:1, took 0.506674 s
[['hadoop'], ['python'], ['spark']]
```

2. flatmap() : It doesn't give individual elements

```
>>> wordsWithFlatMap = lines.flatMap(lambda line: line.split(" "))
>>> wordsWithFlatMap.collect()
17/12/27 09:41:47 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 09:41:47 INFO scheduler.DAGScheduler: Got job 1 (collect at <stdin>:1) with 1 output
partitions
17/12/27 09:41:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all
completed, from pool
['hadoop', 'python', 'spark']
```

3. filter() : It is widely used

Here file indicates the local file system

```
>>> emp_details = sc.textFile("file:/home/cloudera/employeeDataSet.csv")
17/12/27 11:52:57 INFO storage.MemoryStore: ensureFreeSpace(187856) called with curMem=253036,
maxMem=560497950
17/12/27 11:52:57 INFO storage.MemoryStore: Block broadcast_2 stored as values in memory
(estimated size 183.5 KB, free 534.1 MB)
17/12/27 11:52:57 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=440892,
maxMem=560497950
```

APACHE SPARK

17/12/27 11:52:57 INFO storage.MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 20.7 KB, free 534.1 MB)
17/12/27 11:52:57 INFO storage.BlockManagerInfo: Added broadcast_2_piece0 in memory on localhost:39841 (size: 20.7 KB, free: 534.5 MB)
17/12/27 11:52:57 INFO spark.SparkContext: Created broadcast 2 from textFile at NativeMethodAccessorImpl.java:-2
>>> rows = emp_details.map(lambda line: line.split(","))
>>> rows.collect()
17/12/27 11:53:12 INFO mapred.FileInputFormat: Total input paths to process : 1
17/12/27 11:53:12 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 11:53:13 INFO scheduler.DAGScheduler: ResultStage 0 (collect at <stdin>:1) finished in 0.834 s
17/12/27 11:53:13 INFO scheduler.DAGScheduler: Job 0 finished: collect at <stdin>:1, took 1.105805 s
output
[[u''], [u'Year', u'First Name', u'Country', u'Sex', u'Age'], [u'2017', u'NIKHIL', u'INDIA', u'M', u'23'], [u'2017', u'SHIVA', u'INDIA', u'M', u'23'], [u'2017', u'KAUSHIK', u'INDIA', u'M', u'28'], [u'2017', u'JACK', u'USA', u'M', u'35'], [u'2017', u'JULY', u'USA', u'F', u'28']]

Filter: It is widely used
>>> rows.filter(lambda line: "NIKHIL" in line).collect()
17/12/27 11:56:28 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 11:56:28 INFO scheduler.DAGScheduler: Got job 1 (collect at <stdin>:1) with 1 output partitions
17/12/27 11:56:28 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
output
[[u'2017', u'NIKHIL', u'INDIA', u'M', u'23']]

4. mapPartitions() : It is an alternative used for map and foreach

>>> rdd =sc.parallelize([1,2,3,4])
>>> def f(iterator): yield sum(iterator)
...
>>> rdd.mapPartitions(f).collect()
17/12/27 12:05:39 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 12:05:39 INFO scheduler.DAGScheduler: Final stage: ResultStage 2(collect at <stdin>:1)
17/12/27 12:05:39 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 12:05:39 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 12:05:39 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
output
[10]

Or

APACHE SPARK

Here we set the partition to 2 i.e., we can have the result in 2 partitions

```
>>> rdd =sc.parallelize([1,2,3,4],2)
>>> def f(iterator): yield sum(iterator)
...
>>> rdd.mapPartitions(f).collect()
17/12/27 12:08:53 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 12:08:53 INFO scheduler.DAGScheduler: Final stage: ResultStage 3(collect at <stdin>:1)
17/12/27 12:08:53 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 12:08:53 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 12:08:53 INFO scheduler.DAGScheduler: ResultStage 3 (collect at <stdin>:1) finished in 0.114 s
17/12/27 12:08:53 INFO scheduler.DAGScheduler: Job 3 finished: collect at <stdin>:1, took 0.148059 s
output
[3, 7]
```

i.e., 1+2 =3 3+4 = 7

5. mapPartitionsWithIndex() : it is similar to mapPartitions but has an in-built feature of indexing
Here we used range instead of declaring 1,2,3,4,5,6,7,8,9,10 with 4 partitions
+str is used for type conversion

```
>>> parallel = sc.parallelize(range(1,10),4)
>>> def show(index, iterator): yield 'index: '+str(index)+" values: "+str(list(iterator))
...
>>> parallel.mapPartitionsWithIndex(show).collect()
17/12/27 12:23:38 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 12:23:38 INFO scheduler.DAGScheduler: Final stage: ResultStage 6(collect at <stdin>:1)
17/12/27 12:23:38 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 12:23:38 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 12:23:38 INFO scheduler.DAGScheduler: ResultStage 6 (collect at <stdin>:1) finished in 0.236 s
17/12/27 12:23:38 INFO scheduler.DAGScheduler: Job 6 finished: collect at <stdin>:1, took 0.265357 s
['index: 0 values: [1, 2]', 'index: 1 values: [3, 4]', 'index: 2 values: [5, 6]', 'index: 3 values: [7, 8, 9]']
```

6. sample():

7. union(): clubs 2 datasets.

```
>>> one = sc.parallelize(range(1,5))
>>> two = sc.parallelize(range(6,10))
>>> one.union(two).collect()
17/12/27 12:29:42 INFO spark.SparkContext: Starting job: collect at <stdin>:1
```

APACHE SPARK

17/12/27 12:29:42 INFO scheduler.DAGScheduler: Final stage: ResultStage 7(collect at <stdin>:1)
17/12/27 12:29:42 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 12:29:42 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 12:29:42 INFO scheduler.DAGScheduler: ResultStage 7 (collect at <stdin>:1) finished in 0.047 s
17/12/27 12:29:42 INFO scheduler.DAGScheduler: Job 7 finished: collect at <stdin>:1, took 0.095138 s
[1, 2, 3, 4, 6, 7, 8, 9]

8. intersection():

>>> one = sc.parallelize(range(1,5))
>>> two = sc.parallelize(range(3,10))
>>> one.intersection(two).collect()
17/12/27 12:33:35 INFO python.PythonRDD: Times: total = 43, boot = -52718, init = 52760, finish = 1
17/12/27 12:33:35 INFO python.PythonRDD: Times: total = 48, boot = -52786, init = 52833, finish = 1
17/12/27 12:33:35 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 10.0 (TID 24, localhost, partition 1,PROCESS_LOCAL, 2201 bytes)
17/12/27 12:33:35 INFO executor.Executor: Running task 1.0 in stage 10.0 (TID 24)
17/12/27 12:33:35 INFO scheduler.DAGScheduler: ResultStage 11 (collect at <stdin>:1) finished in 0.127 s
17/12/27 12:33:35 INFO scheduler.DAGScheduler: Job 9 finished: collect at <stdin>:1, took 0.411425 s
[4, 3]

9. distinct():

>>> one = sc.parallelize(range(1,5))
>>> two = sc.parallelize(range(3,10))
>>> one.union(two).distinct().collect()
17/12/27 12:37:06 INFO python.PythonRDD: Times: total = 11, boot = 3, init = 4, finish = 4
17/12/27 12:37:06 INFO python.PythonRDD: Times: total = 39, boot = 4, init = 17, finish = 18
17/12/27 12:37:06 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 12.0 (TID 28, localhost, partition 1,PROCESS_LOCAL, 2201 bytes)
17/12/27 12:37:06 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 13.0, whose tasks have all completed, from pool
[8, 2, 4, 6, 1, 3, 9, 5, 7]

10. groupByKey() : the dataset should have key value pairs in the dataset

>>> emp_details = sc.textFile("file:/home/cloudera/employeeDataSet.csv")
17/12/27 12:44:47 INFO storage.MemoryStore: ensureFreeSpace(187856) called with curMem=585366, maxMem=560497950
17/12/27 12:44:47 INFO storage.MemoryStore: Block broadcast_17 stored as values in memory (estimated size 183.5 KB, free 533.8 MB)
17/12/27 12:44:47 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=773222, maxMem=560497950

APACHE SPARK

17/12/27 12:44:47 INFO storage.MemoryStore: Block broadcast_17_piece0 stored as bytes in memory (estimated size 20.7 KB, free 533.8 MB)
17/12/27 12:44:47 INFO storage.BlockManagerInfo: Added broadcast_17_piece0 in memory on localhost:39841 (size: 20.7 KB, free: 534.4 MB)
17/12/27 12:44:47 INFO spark.SparkContext: Created broadcast 17 from textFile at NativeMethodAccessorImpl.java:-2
>>> rows = emp_details.map(lambda line: line.split(","))
>>> namesToCountries.map(lambda x : {x[0]: list(x[1])}).collect()

11. reduceByKey(): it operates on key value pairs

>>> emp_details = sc.textFile("file:/home/cloudera/employeeDataSet.csv")
17/12/27 13:04:28 INFO storage.MemoryStore: ensureFreeSpace(187856) called with curMem=907691, maxMem=560497950
17/12/27 13:04:28 INFO storage.MemoryStore: Block broadcast_22 stored as values in memory (estimated size 183.5 KB, free 533.5 MB)
17/12/27 13:04:28 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=1095547, maxMem=560497950
17/12/27 13:04:28 INFO storage.MemoryStore: Block broadcast_22_piece0 stored as bytes in memory (estimated size 20.7 KB, free 533.5 MB)
17/12/27 13:04:28 INFO storage.BlockManagerInfo: Added broadcast_22_piece0 in memory on localhost:39841 (size: 20.7 KB, free: 534.4 MB)
17/12/27 13:04:28 INFO spark.SparkContext: Created broadcast 22 from textFile at NativeMethodAccessorImpl.java:-2
>>> rows = emp_details.map(lambda line: line.split(","))
>>> filtered_rows = emp_details.filter(lambda line: "Age" not in line).map(lambda line: line.split(","))
>>> filtered_rows.map(lambda n: (str(n[1]), int(n[4]))).reduceByKey(lambda v1,v2: v1 + v2).collect()

12. aggregateByKey():
13. sortByKey()
14. join() : By default, it performs inner join
Here we are adding 1
>>> names1 = sc.parallelize(("Nikhil", "Shiva", "Dilip")).map(lambda a: (a,1))
>>> names2 = sc.parallelize(("Apple", "Shiva", "Sony")).map(lambda a: (a,1))
>>> names1.join(names2).collect()
17/12/27 13:28:54 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 13:28:54 INFO scheduler.DAGScheduler: Registering RDD 23 (join at <stdin>:1)
17/12/27 13:28:54 INFO scheduler.DAGScheduler: Final stage: ResultStage 7(collect at <stdin>:1)
17/12/27 13:28:54 INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 6)
17/12/27 13:28:54 INFO scheduler.DAGScheduler: Missing parents: List(ShuffleMapStage 6)
17/12/27 13:28:54 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 7.0 (TID 6) in 66 ms on localhost (2/2)

APACHE SPARK

17/12/27 13:28:54 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks have all completed, from pool
[('Shiva', (1, 1))]

>>> names1.leftOuterJoin(names2).collect()
17/12/27 13:30:55 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 13:30:55 INFO scheduler.DAGScheduler: Registering RDD 31 (leftOuterJoin at <stdin>:1)
17/12/27 13:30:55 INFO scheduler.DAGScheduler: Final stage: ResultStage 9(collect at <stdin>:1)
17/12/27 13:30:55 INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 8)
17/12/27 13:30:55 INFO scheduler.DAGScheduler: Missing parents: List(ShuffleMapStage 8)
17/12/27 13:30:55 INFO scheduler.DAGScheduler: ResultStage 9 (collect at <stdin>:1) finished in 0.105 s
17/12/27 13:30:55 INFO scheduler.DAGScheduler: Job 4 finished: collect at <stdin>:1, took 0.258792 s
[('Shiva', (1, 1)), ('Nikhil', (1, None)), ('Dilip', (1, None))]

>>> names1.rightOuterJoin(names2).collect()
17/12/27 13:32:56 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 13:32:56 INFO scheduler.DAGScheduler: Registering RDD 39 (rightOuterJoin at <stdin>:1)
17/12/27 13:32:56 INFO scheduler.DAGScheduler: Final stage: ResultStage 11(collect at <stdin>:1)
17/12/27 13:32:56 INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 10)
17/12/27 13:32:56 INFO scheduler.DAGScheduler: Missing parents: List(ShuffleMapStage 10)
17/12/27 13:32:56 INFO scheduler.DAGScheduler: Job 5 finished: collect at <stdin>:1, took 0.347766 s
[('Shiva', (1, 1)), ('Sony', (None, 1)), ('Apple', (None, 1))]
>>>
15. mapValues()

Actions

Return a value
Ex: reduce(), collect(), count(), first(), take(), foreach(), countByKey(), takeSample(), savaAsTextFile()

1. first() : It returns the first element of the dataset.

>>> names = sc.parallelize(["abc", "def","ghi"])
>>> names.first()
...

APACHE SPARK

17/12/27 15:26:36 INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:361
17/12/27 15:26:36 INFO scheduler.DAGScheduler: Got job 0 (runJob at PythonRDD.scala:361) with 1 output partitions
17/12/27 15:26:36 INFO scheduler.DAGScheduler: Final stage: ResultStage 0(runJob at PythonRDD.scala:361)
17/12/27 15:26:36 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:26:36 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:26:36 INFO scheduler.DAGScheduler: Submitting ResultStage 0 (PythonRDD[1] at RDD at PythonRDD.scala:43), which has no missing parents
17/12/27 15:26:38 INFO scheduler.DAGScheduler: ResultStage 0 (runJob at PythonRDD.scala:361) finished in 0.845 s
17/12/27 15:26:38 INFO scheduler.DAGScheduler: Job 0 finished: runJob at PythonRDD.scala:361, took 1.659590 s
'abc'

2. collect() : Returns all the elements of the dataset

>>> names.collect()
17/12/27 15:30:17 INFO spark.ContextCleaner: Cleaned accumulator 2
17/12/27 15:30:17 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 15:30:17 INFO scheduler.DAGScheduler: Final stage: ResultStage 1(collect at <stdin>:1)
17/12/27 15:30:17 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:30:17 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:30:17 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 37 ms on localhost (1/1)
17/12/27 15:30:17 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
['abc', 'def', 'ghi']

3. count(): Returns the no. of elements in the dataset.

>>> names.count()
17/12/27 15:32:36 INFO spark.SparkContext: Starting job: count at <stdin>:1
17/12/27 15:32:36 INFO scheduler.DAGScheduler: Got job 2 (count at <stdin>:1) with 1 output partitions
17/12/27 15:32:36 INFO scheduler.DAGScheduler: Final stage: ResultStage 2(count at <stdin>:1)
17/12/27 15:32:36 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:32:36 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:32:36 INFO scheduler.DAGScheduler: Submitting ResultStage 2 (PythonRDD[2] at count at <stdin>:1), which has no missing parents
17/12/27 15:32:36 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2.0 (TID 2) in 101 ms on localhost (1/1)
17/12/27 15:32:36 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool

APACHE SPARK

4. take(): It takes arguments. If you pass 1 as argument, it displays the first element of the dataset.

```
>>> names.take(1)
17/12/27 15:33:53 INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:361
17/12/27 15:33:53 INFO scheduler.DAGScheduler: Got job 3 (runJob at PythonRDD.scala:361) with 1
output partitions
17/12/27 15:33:53 INFO scheduler.DAGScheduler: Final stage: ResultStage 3(runJob at
PythonRDD.scala:361)
17/12/27 15:33:53 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:33:53 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:33:53 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all
completed, from pool
['abc']
```

5. takeSample(): it works randomly. It at least takes 3 arguments. One argument should be true/false
   True – gives non-unique set of elements
   False – gives unique set of elements

```
>>> names = sc.parallelize(["abc", "def","ghi", "jki", "mno", "pqr", "stu", "vwx", "yz"])
>>> names.takeSample(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: takeSample() takes at least 3 arguments (2 given)
>>> names.takeSample(True,2)
17/12/27 15:37:40 INFO spark.SparkContext: Starting job: takeSample at <stdin>:1
17/12/27 15:37:40 INFO scheduler.DAGScheduler: Final stage: ResultStage 4(takeSample at <stdin>:1)
17/12/27 15:37:40 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:37:40 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:37:40 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all
completed, from pool
['jki', 'stu']
```

```
>>> names.takeSample(False,2)
17/12/27 15:39:59 INFO spark.SparkContext: Starting job: takeSample at <stdin>:1
17/12/27 15:39:59 INFO scheduler.DAGScheduler: Got job 6 (takeSample at <stdin>:1) with 1 output
partitions
17/12/27 15:39:59 INFO scheduler.DAGScheduler: Final stage: ResultStage 6(takeSample at <stdin>:1)
17/12/27 15:39:59 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:39:59 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:39:59 INFO scheduler.DAGScheduler: ResultStage 7 (takeSample at <stdin>:1) finished in
0.055 s
```

APACHE SPARK

17/12/27 15:39:59 INFO scheduler.DAGScheduler: Job 7 finished: takeSample at <stdin>:1, took 0.085366 s
['ghi', 'def']

6. countByKey(): it counts the total no of elements in each key and returns the value in dictionary format.

```
>>> names1 = sc.parallelize (["Nikhil", "hadoop", "Spark", "Flume", "Scoop", "Scala", "Nikhil", "hadoop"])
>>> names1.map(lambda k: (k,1)).countByKey( ).items()
17/12/27 15:47:35 INFO spark.SparkContext: Starting job: countByKey at <stdin>:1
17/12/27 15:47:35 INFO scheduler.DAGScheduler: Got job 8 (countByKey at <stdin>:1) with 1 output partitions
17/12/27 15:47:35 INFO scheduler.DAGScheduler: Final stage: ResultStage 8(countByKey at <stdin>:1)
17/12/27 15:47:35 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:47:35 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:47:36 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 8.0 (TID 8) in 60 ms on localhost (1/1)
17/12/27 15:47:36 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all completed, from pool
[('Flume', 1), ('Nikhil', 2), ('Scala', 1), ('hadoop', 2), ('Scoop', 1), ('Spark', 1)]
```

7. reduce():

it performs concatenation

```
>>> names1 = sc.parallelize (["Nikhil", "hadoop", "Spark", "Flume", "Scoop", "Scala", "Nikhil", "hadoop"])
>>> print names1.reduce(lambda t1,t2: t1+t2)
17/12/27 15:50:34 INFO spark.SparkContext: Starting job: reduce at <stdin>:1
17/12/27 15:50:34 INFO scheduler.DAGScheduler: Got job 9 (reduce at <stdin>:1) with 1 output partitions
17/12/27 15:50:34 INFO scheduler.DAGScheduler: Final stage: ResultStage 9(reduce at <stdin>:1)
17/12/27 15:50:34 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:50:34 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:50:34 INFO scheduler.DAGScheduler: ResultStage 9 (reduce at <stdin>:1) finished in 0.042 s
17/12/27 15:50:34 INFO scheduler.DAGScheduler: Job 9 finished: reduce at <stdin>:1, took 0.078288 s
NikhilhadoopSparkFlumeScoopScalaNikhilhadoop
```

each individual element in the dataset has its own length

```
>>> names1 = sc.parallelize (["Nikhil", "hadoop", "Spark", "Flume", "Scoop", "Scala", "Nikhil","hadoop"]).map(lambda i: [i, len(i)])
>>> names1.collect()
17/12/27 15:51:49 INFO spark.SparkContext: Starting job: collect at <stdin>:1
```

APACHE SPARK

```
17/12/27 15:51:49 INFO scheduler.DAGScheduler: Final stage: ResultStage 10(collect at <stdin>:1)
17/12/27 15:51:49 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:51:49 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:51:49 INFO scheduler.DAGScheduler: ResultStage 10 (collect at <stdin>:1) finished in 0.036
s
17/12/27 15:51:49 INFO scheduler.DAGScheduler: Job 10 finished: collect at <stdin>:1, took 0.080020 s
[['Nikhil', 6], ['hadoop', 6], ['Spark', 5], ['Flume', 5], ['Scoop', 5], ['Scala', 5], ['Nikhil', 6], ['hadoop', 6]]
>>>
```

total length of the concatenation elements

```
>>> names1.flatMap(lambda t: [t[1]]).reduce(lambda t1,t2: t1+t2)
17/12/27 15:57:36 INFO spark.SparkContext: Starting job: reduce at <stdin>:1
17/12/27 15:57:36 INFO scheduler.DAGScheduler: Final stage: ResultStage 11(reduce at <stdin>:1)
17/12/27 15:57:36 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 15:57:36 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 15:57:37 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 11.0 (TID 11) in 36 ms on
localhost (1/1)
17/12/27 15:57:37 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 11.0, whose tasks have all
completed, from pool
44
```

8. savaAsTextFile

```
>>> counts.saveAsTextFile("file:/home/cloudera/SparkSamples/wordCountOutput")
17/12/27 16:45:22 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/27 16:45:22 INFO spark.SparkContext: Starting job: saveAsTextFile at
NativeMethodAccessorImpl.java:-2
17/12/27 16:45:22 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 0 is 143
bytes
17/12/27 16:45:23 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0 (TID 3) in 456 ms on
localhost (1/1)
17/12/27 16:45:23 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all
completed, from pool
```

APACHE SPARK

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/SparkSamples
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2017-12-27 09:10 /user/SparkSamples/employeeDataSet.csv
-rw-r--r--   1 cloudera supergroup          0 2017-12-27 16:12 /user/SparkSamples/wordCount.txt
[cloudera@quickstart ~]$ cd wordCountOutput
bash: cd: wordCountOutput: No such file or directory
[cloudera@quickstart ~]$ cd SparkSamples
[cloudera@quickstart SparkSamples]$ ls
wordCountOutput
[cloudera@quickstart SparkSamples]$ cat wordCOuntOutput
cat: wordCOuntOutput: No such file or directory
[cloudera@quickstart SparkSamples]$ cat wordCountOutput
cat: wordCountOutput: Is a directory
[cloudera@quickstart SparkSamples]$ cd wordCountOutput
[cloudera@quickstart wordCountOutput]$ ls
part-00000  _SUCCESS
[cloudera@quickstart wordCountOutput]$ cat part-00000
(u'', 1)
(u'EveryOne', 1)
(u'Welcome', 1)
(u'Hello', 1)
(u'1', 1)
(u'to', 1)
(u'Tutorials', 1)
(u'Spark', 1)
(u'Class', 1)
[cloudera@quickstart wordCountOutput]$
```

[cloudera@quickstart ~]$ hadoop fs -ls /user/SparkSamples
Found 1 items
-rw-r--r-- 1 cloudera supergroup 0 2017-12-27 09:10 /user/SparkSamples/employeeDataSet.csv
[cloudera@quickstart ~]$ hadoop fs -touchz /user/SparkSamples/wordCount.txt
[cloudera@quickstart ~]$ vim wordCount.txt
[cloudera@quickstart ~]$ cat wordCount.txt

Hello EveryOne
Welcome to
Spark Tutorials
Class 1

>>> sc.textFile("file:/home/cloudera/wordCount.txt").collect()

Word Count Use case

>>> text_file = sc.textFile("file:/home/cloudera/wordCount.txt").collect()
17/12/27 16:20:22 INFO storage.MemoryStore: ensureFreeSpace(187856) called with curMem=476539, maxMem=560497950

APACHE SPARK

17/12/27 16:20:22 INFO storage.MemoryStore: Block broadcast_17 stored as values in memory (estimated size 183.5 KB, free 533.9 MB)
17/12/27 16:20:22 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=664395, maxMem=560497950
17/12/27 16:20:22 INFO storage.MemoryStore: Block broadcast_17_piece0 stored as bytes in memory (estimated size 20.7 KB, free 533.9 MB)
17/12/27 16:20:22 INFO storage.BlockManagerInfo: Added broadcast_17_piece0 in memory on localhost:58556 (size: 20.7 KB, free: 534.5 MB)
17/12/27 16:20:22 INFO spark.SparkContext: Created broadcast 17 from textFile at NativeMethodAccessorImpl.java:-2
17/12/27 16:20:22 INFO mapred.FileInputFormat: Total input paths to process : 1
17/12/27 16:20:22 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Got job 14 (collect at <stdin>:1) with 1 output partitions
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Final stage: ResultStage 14(collect at <stdin>:1)
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Missing parents: List()
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Submitting ResultStage 14 (MapPartitionsRDD[26] at textFile at NativeMethodAccessorImpl.java:-2), which has no missing parents
17/12/27 16:20:22 INFO storage.MemoryStore: ensureFreeSpace(3144) called with curMem=685628, maxMem=560497950
17/12/27 16:20:22 INFO storage.MemoryStore: Block broadcast_18 stored as values in memory (estimated size 3.1 KB, free 533.9 MB)
17/12/27 16:20:22 INFO storage.MemoryStore: ensureFreeSpace(1814) called with curMem=688772, maxMem=560497950
17/12/27 16:20:22 INFO storage.MemoryStore: Block broadcast_18_piece0 stored as bytes in memory (estimated size 1814.0 B, free 533.9 MB)
17/12/27 16:20:22 INFO storage.BlockManagerInfo: Added broadcast_18_piece0 in memory on localhost:58556 (size: 1814.0 B, free: 534.5 MB)
17/12/27 16:20:22 INFO spark.SparkContext: Created broadcast 18 from broadcast at DAGScheduler.scala:861
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 14 (MapPartitionsRDD[26] at textFile at NativeMethodAccessorImpl.java:-2)
17/12/27 16:20:22 INFO scheduler.TaskSchedulerImpl: Adding task set 14.0 with 1 tasks
17/12/27 16:20:22 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 14.0 (TID 14, localhost, partition 0,PROCESS_LOCAL, 2142 bytes)
17/12/27 16:20:22 INFO executor.Executor: Running task 0.0 in stage 14.0 (TID 14)
17/12/27 16:20:22 INFO rdd.HadoopRDD: Input split: file:/home/cloudera/wordCount.txt:0+51
17/12/27 16:20:22 INFO executor.Executor: Finished task 0.0 in stage 14.0 (TID 14). 2105 bytes result sent to driver
17/12/27 16:20:22 INFO scheduler.DAGScheduler: ResultStage 14 (collect at <stdin>:1) finished in 0.031 s
17/12/27 16:20:22 INFO scheduler.DAGScheduler: Job 14 finished: collect at <stdin>:1, took 0.060263 s
17/12/27 16:20:22 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 14.0 (TID 14) in 32 ms on localhost (1/1)
17/12/27 16:20:22 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 14.0, whose tasks have all completed, from pool

APACHE SPARK

```
>>> text_file = sc.textFile("file:/home/cloudera/wordCount.txt")
17/12/27 16:38:36 INFO storage.MemoryStore: ensureFreeSpace(92440) called with curMem=144316, maxMem=560497950
17/12/27 16:38:36 INFO storage.MemoryStore: Block broadcast_2 stored as values in memory (estimated size 90.3 KB, free 534.3 MB)
17/12/27 16:38:36 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=236756, maxMem=560497950
17/12/27 16:38:36 INFO storage.MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 20.7 KB, free 534.3 MB)
17/12/27 16:38:36 INFO storage.BlockManagerInfo: Added broadcast_2_piece0 in memory on localhost:36797 (size: 20.7 KB, free: 534.5 MB)
17/12/27 16:38:36 INFO spark.SparkContext: Created broadcast 2 from textFile at NativeMethodAccessorImpl.java:-2
```

```
>>> counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word,1)).reduceByKey(lambda a, b: a + b)
17/12/27 16:40:49 INFO mapred.FileInputFormat: Total input paths to process : 1
```

```
>>> counts.collect()
17/12/27 16:41:42 INFO spark.SparkContext: Starting job: collect at <stdin>:1
17/12/27 16:41:42 INFO scheduler.DAGScheduler: Registering RDD 5 (reduceByKey at <stdin>:1)
17/12/27 16:41:42 INFO scheduler.DAGScheduler: Got job 1 (collect at <stdin>:1) with 1 output partitions
17/12/27 16:41:44 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
[(u'', 1), (u'EveryOne', 1), (u'Welcome', 1), (u'Hello', 1), (u'1', 1), (u'to', 1), (u'Tutorials', 1), (u'Spark', 1), (u'Class', 1)]
>>>
```

Here comes the concept of saveAstextFile concept. The output from the word count program is now saved as a text file.

```
>>> counts.saveAsTextFile("file:/home/cloudera/SparkSamples/wordCountOutput")
17/12/27 16:45:22 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/27 16:45:22 INFO spark.SparkContext: Starting job: saveAsTextFile at NativeMethodAccessorImpl.java:-2
17/12/27 16:45:22 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 0 is 143 bytes
17/12/27 16:45:22 INFO scheduler.DAGScheduler: Got job 2 (saveAsTextFile at NativeMethodAccessorImpl.java:-2) with 1 output partitions
17/12/27 16:45:22 INFO scheduler.DAGScheduler: Submitting ResultStage 4 (MapPartitionsRDD[11] at saveAsTextFile at NativeMethodAccessorImpl.java:-2), which has no missing parents
17/12/27 16:45:22 INFO storage.MemoryStore: ensureFreeSpace(138808) called with curMem=279625, maxMem=560497950
17/12/27 16:45:22 INFO storage.MemoryStore: Block broadcast_5 stored as values in memory (estimated size 135.6 KB, free 534.1 MB)
```

APACHE SPARK

17/12/27 16:45:22 INFO storage.MemoryStore: ensureFreeSpace(48658) called with curMem=418433, maxMem=560497950
17/12/27 16:45:22 INFO storage.MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 47.5 KB, free 534.1 MB)
17/12/27 16:45:23 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/SparkSamples
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2017-12-27 09:10 /user/SparkSamples/employeeDataSet.csv
-rw-r--r--   1 cloudera supergroup          0 2017-12-27 16:12 /user/SparkSamples/wordCount.txt
[cloudera@quickstart ~]$ cd wordCountOutput
bash: cd: wordCountOutput: No such file or directory
[cloudera@quickstart ~]$ cd SparkSamples
[cloudera@quickstart SparkSamples]$ ls
wordCountOutput
[cloudera@quickstart SparkSamples]$ cat wordCOuntOutput
cat: wordCOuntOutput: No such file or directory
[cloudera@quickstart SparkSamples]$ cat wordCountOutput
cat: wordCountOutput: Is a directory
[cloudera@quickstart SparkSamples]$ cd wordCountOutput
[cloudera@quickstart wordCountOutput]$ ls
part-00000  _SUCCESS
[cloudera@quickstart wordCountOutput]$ cat part-00000
(u'', 1)
(u'EveryOne', 1)
(u'Welcome', 1)
(u'Hello', 1)
(u'1', 1)
(u'to', 1)
(u'Tutorials', 1)
(u'Spark', 1)
(u'Class', 1)
[cloudera@quickstart wordCountOutput]$
```

UBER Use case

[cloudera@quickstart]$ hadoop fs -ls /user/SparkSamples
Found 3 items
-rw-r--r-- 1 cloudera supergroup 0 2017-12-27 09:10 /user/SparkSamples/employeeDataSet.csv
-rw-r--r-- 1 cloudera supergroup 0 2017-12-27 17:02 /user/SparkSamples/uberDataSet.csv
-rw-r--r-- 1 cloudera supergroup 0 2017-12-27 16:12 /user/SparkSamples/wordCount.txt
[cloudera@quickstart]$ vim uberDataSet.csv
[cloudera@quickstart]$ cat uberDataSet.csv

>>> uber = sc.textFile("file:/home/cloudera/uberDataSet.csv")

APACHE SPARK

17/12/27 17:45:36 INFO storage.MemoryStore: ensureFreeSpace(187856) called with curMem=885269, maxMem=560497950
17/12/27 17:45:36 INFO storage.MemoryStore: Block broadcast_8 stored as values in memory (estimated size 183.5 KB, free 533.5 MB)
17/12/27 17:45:36 INFO storage.MemoryStore: ensureFreeSpace(21233) called with curMem=1073125, maxMem=560497950
17/12/27 17:45:36 INFO storage.MemoryStore: Block broadcast_8_piece0 stored as bytes in memory (estimated size 20.7 KB, free 533.5 MB)
17/12/27 17:45:36 INFO storage.BlockManagerInfo: Added broadcast_8_piece0 in memory on localhost:36797 (size: 20.7 KB, free: 534.4 MB)
17/12/27 17:45:36 INFO spark.SparkContext: Created broadcast 8 from textFile at NativeMethodAccessorImpl.java:-2

#total count

>>> uber.count()
17/12/27 17:47:06 INFO spark.SparkContext: Starting job: count at <stdin>:1
17/12/27 17:47:06 INFO scheduler.DAGScheduler: Got job 4 (count at <stdin>:1) with 1 output partitions
17/12/27 17:47:06 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all completed, from pool
1439

#to see the header

>>> uber.first()
17/12/27 17:47:30 INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:361
17/12/27 17:47:30 INFO scheduler.DAGScheduler: Got job 5 (runJob at PythonRDD.scala:361) with 1 output partitions
17/12/27 17:47:30 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks have all completed, from pool
u'"Date/Time","Lat","Lon","Base"'

Connecting HIVE from Spark

To connect Hive from Spark, firstly we need to copy the hive config path from hive to spark

Hive path

APACHE SPARK

```
[cloudera@quickstart ~]$ cd /etc/hive/conf
[cloudera@quickstart conf]$ ls -lrt
total 20
-rw-r--r-- 1 root root 3505 Nov  9  2015 hive-log4j.properties
-rw-r--r-- 1 root root 2662 Nov  9  2015 hive-exec-log4j.properties
-rw-r--r-- 1 root root 2378 Nov  9  2015 hive-env.sh.template
-rw-r--r-- 1 root root 1139 Nov  9  2015 beeline-log4j.properties.template
-rw-rw-r-- 1 root root 1937 Nov 18  2015 hive-site.xml
[cloudera@quickstart conf]$
```

Copy the hive-site.xml to spark path. So, we are creating the soft link for that purpose. Soft link, if any changes done in the actual file, the changes are also reflected in the soft link.
If permission is denied, use the command **sudo**

```
[cloudera@quickstart conf]$ ln -s /etc/hive/conf/hive-site.xml /etc/spark/conf
ln: creating symbolic link `/etc/spark/conf/hive-site.xml': Permission denied
[cloudera@quickstart conf]$ sudo ln -s /etc/hive/conf/hive-site.xml /etc/spark/conf
[cloudera@quickstart conf]$ cd /etc/spark/conf
[cloudera@quickstart conf]$ ls -lrt
total 48
-rwxr-xr-x 1 root root 3418 Nov  9  2015 spark-env.sh.template
-rw-r--r-- 1 root root  507 Nov  9  2015 spark-defaults.conf.template
-rw-r--r-- 1 root root   80 Nov  9  2015 slaves.template
-rw-r--r-- 1 root root 5886 Nov  9  2015 metrics.properties.template
-rw-r--r-- 1 root root  949 Nov  9  2015 log4j.properties.template
-rw-r--r-- 1 root root  303 Nov  9  2015 fairscheduler.xml.template
-rw-r--r-- 1 root root  202 Nov  9  2015 docker.properties.template
-rwxr-xr-x 1 root root 5238 Nov  9  2015 spark-env.sh
-rw-r--r-- 1 root root  507 Nov  9  2015 spark-defaults.conf
-rw-rw-r-- 1 root root   92 Nov 18  2015 slaves
lrwxrwxrwx 1 root root   28 Dec 29 07:51 hive-site.xml -> /etc/hive/conf/hive-site.xml
[cloudera@quickstart conf]$
```

Example

sqlContext.sql(“select * from sample_07”).collect()

Here sample_07 is the table

APACHE SPARK

DataFrames

DataFrame is another API which is more powerful than RDD
DataFrame is a distributed collection of data organized into named columns. It has same properties as RDD.
DataFrame can be capable of handling very huge set of data.

```
//converting DF to RDD
rddVar = dfVar.rdd

//converting RDD to DF
dfVar = sqlContext.createDataframe(rddVar)
```

Important classes of Spark SQL and DataFrames

- **pyspark.sql.SparkSession** Main entry point for DataFrames & functionality
- **pyspark.sql.DataFrame** Distributed collection of data grouped into named columns
- **pyspark.sql.Column** Column expression in a DataFrame
- **pyspark.sql.Row** Row of data in a DataFrame
- **pyspark.sql.GroupedData** Aggregation methods, returned by DataFrame.groupBy()
- **pyspark.sql.DataFrameNaFunctions** Methods for handling missing data (null values)
- **pyspark.sql.DataFrameStatFunctions** Methods for statistics functionality
- **pyspark.sql.functions** List of built-in functions available for DataFrame
- **pyspark.sql.types** List of datatypes available
- **pyspark.sql.Window** For working with windows functions

DataFrame operations

1. printSchema
2. head
3. show
4. describe
5. count
6. select
7. distinct
8. crosstab : calculate pair wise frequency of categorical columns
9. dropDuplicates
10. dropna : drop all rows with null values
11. fillna : replacing null values in FileSystem with constant
12. filter
13. groupby
14. sample : creating child DF from parent DF

Transformation operations on DataFrame columns

APACHE SPARK

15. orderBy : sorting
16. withColumn: adding new column in DF

input for DataFrame is
it can be a file, or a hive table, it can be some external RDBMS, even the input can be RDD

```
>>> from pyspark.sql import Row
>>> var = [('Nikhil',1000),('Dilip', 2000),('Shiva',3000),('Nithesh',4000),('Nikhil',1000)]
>>> rdd = sc.parallelize(var)
>>> rdd.collect()
>>> rddvar = rdd.map(lambda x: Row(name=x[0], salary=x[1]))          //creating schema for name,sal

//converting rddvar into data frame

>>> dfVar = sqlContext.createDataFrame(rddvar)

18/01/03 12:08:26 INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:361
18/01/03 12:08:26 INFO scheduler.DAGScheduler: Got job 2 (runJob at PythonRDD.scala:361) with 1
output partitions
18/01/03 12:08:30 INFO session.SessionState: Created HDFS directory: file:/tmp/spark-4695b441-aca1-
4e0e-9921-0237d4b4fdd9/scratch/cloudera
18/01/03 12:08:30 INFO session.SessionState: Created local directory: /tmp/e2df694a-4551-42ac-9c37-
439190837711_resources
18/01/03 12:08:30 INFO client.ClientWrapper: Inspected Hadoop version: 2.6.0-cdh5.5.0
18/01/03 12:08:30 INFO client.ClientWrapper: Loaded org.apache.hadoop.hive.shims.Hadoop23Shims
for Hadoop version 2.6.0-cdh5.5.0

>>> dfVar.printSchema()
root
 |-- name: string (nullable = true)
 |-- salary: long (nullable = true)



>>> dfVar.collect()
18/01/03 12:14:18 INFO spark.SparkContext: Starting job: collect at <stdin>:1
18/01/03 12:14:18 INFO scheduler.DAGScheduler: Got job 3 (collect at <stdin>:1) with 1 output
partitions
18/01/03 12:14:18 INFO scheduler.DAGScheduler: Final stage: ResultStage 3(collect at <stdin>:1)
18/01/03 12:14:18 INFO scheduler.DAGScheduler: Parents of final stage: List()
18/01/03 12:14:18 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all
completed, from pool
[Row(name=u'Nikhil', salary=1000), Row(name=u'Dilip', salary=2000), Row(name=u'Shiva', salary=3000),
Row(name=u'Nithesh', salary=4000), Row(name=u'Nikhil', salary=1000)]
```

APACHE SPARK

```
>>> dfVar.head(3)
18/01/03 12:15:29 INFO spark.SparkContext: Starting job: head at <stdin>:1
18/01/03 12:15:29 INFO scheduler.DAGScheduler: Registering RDD 8 (head at <stdin>:1)
18/01/03 12:15:30 INFO scheduler.DAGScheduler: Job 4 finished: head at <stdin>:1, took 0.450669 s
18/01/03 12:15:30 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 5.0 (TID 5) in 80 ms on
localhost (1/1)
18/01/03 12:15:30 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all
completed, from pool
[Row(name=u'Nikhil', salary=1000), Row(name=u'Dilip', salary=2000), Row(name=u'Shiva', salary=3000)]
```

head is same as show. It only limits the data to be displayed

```
>>> dfVar.show(3)
18/01/03 12:16:56 INFO spark.SparkContext: Starting job: showString at
NativeMethodAccessorImpl.java:-2
18/01/03 12:16:56 INFO scheduler.DAGScheduler: Got job 5 (showString at
NativeMethodAccessorImpl.java:-2) with 1 output partitions
18/01/03 12:16:56 INFO scheduler.DAGScheduler: Final stage: ResultStage 6(showString at
NativeMethodAccessorImpl.java:-2)
18/01/03 12:16:56 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all
completed, from pool
+------+------+
|  name|salary|
+------+------+
|Nikhil|  1000|
| Dilip|  2000|
| Shiva|  3000|
+------+------+
only showing top 3 rows
```

```
>>> dfVar.describe()
18/01/03 12:19:10 INFO storage.BlockManagerInfo: Removed broadcast_6_piece0 on localhost:49221
in memory (size: 4.9 KB, free: 534.5 MB)
18/01/03 12:19:10 INFO spark.ContextCleaner: Cleaned accumulator 8
18/01/03 12:19:10 INFO storage.BlockManagerInfo: Removed broadcast_5_piece0 on localhost:49221
in memory (size: 6.3 KB, free: 534.5 MB)
18/01/03 12:19:12 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 8.0 (TID 8) in 420 ms on
localhost (1/1)
18/01/03 12:19:12 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all
completed, from pool
DataFrame[summary: string, salary: string]
```

APACHE SPARK

```
>>> dfVar.describe().show()
18/01/03 14:41:46 INFO spark.SparkContext: Starting job: describe at NativeMethodAccessorImpl.java:-2
18/01/03 14:41:46 INFO scheduler.DAGScheduler: Registering RDD 23 (describe at NativeMethodAccessorImpl.java:-2)
18/01/03 14:41:46 INFO scheduler.DAGScheduler: Got job 7 (describe at NativeMethodAccessorImpl.java:-2) with 1 output partitions
18/01/03 14:41:46 INFO scheduler.DAGScheduler: Final stage: ResultStage 10(describe at NativeMethodAccessorImpl.java:-2)
18/01/03 14:41:47 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 10.0 (TID 10) in 80 ms on localhost (1/1)
18/01/03 14:41:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
+-------+----------------+
|summary|          salary|
+-------+----------------+
|  count|               5|
|   mean|          2200.0|
| stddev|1166.19037896906|
|    min|            1000|
|    max|            4000|
+-------+----------------+
```

The above are the built-in functions

```
>>> dfVar.describe('name').show()
18/01/03 14:45:52 INFO spark.SparkContext: Starting job: describe at NativeMethodAccessorImpl.java:-2
18/01/03 14:45:52 INFO scheduler.DAGScheduler: Registering RDD 32 (describe at NativeMethodAccessorImpl.java:-2)
18/01/03 14:45:54 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 12.0 (TID 12) in 482 ms on localhost (1/1)
18/01/03 14:45:54 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 12.0, whose tasks have all completed, from pool
+-------+-----+
|summary| name|
+-------+-----+
|  count|    5|
|   mean| null|
| stddev| null|
|    min|Dilip|
|    max|Shiva|
```

Mean, stddev will be null and min, max is calculated based on the ASCII value.

APACHE SPARK

```
>>> dfVar.count()
18/01/03 14:49:28 INFO spark.SparkContext: Starting job: count at NativeMethodAccessorImpl.java:-2
18/01/03 14:49:28 INFO scheduler.DAGScheduler: Registering RDD 39 (count at
NativeMethodAccessorImpl.java:-2)
18/01/03 14:49:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 14.0, whose tasks have all
completed, from pool
5
```

#To select the schema. Here we have selected the column name

```
>>> dfVar.select('name').show()
18/01/03 14:50:58 INFO spark.SparkContext: Starting job: showString at
NativeMethodAccessorImpl.java:-2
18/01/03 14:50:58 INFO scheduler.DAGScheduler: Got job 10 (showString at
NativeMethodAccessorImpl.java:-2) with 1 output partitions
18/01/03 14:50:58 INFO scheduler.DAGScheduler: Final stage: ResultStage 15(showString at
NativeMethodAccessorImpl.java:-2)
18/01/03 14:50:58 INFO storage.MemoryStore: Block broadcast_15 stored as values in memory
(estimated size 10.3 KB, free 534.3 MB)
18/01/03 14:50:58 INFO storage.MemoryStore: ensureFreeSpace(6008) called with curMem=232253,
maxMem=560497950
+-------+
|   name|
+-------+
| Nikhil|
|  Dilip|
|  Shiva|
|Nithesh|
| Nikhil|
+-------+
```

#here we have selected the column name as salary

```
dfVar.select('salary').show()
18/01/03 14:52:47 INFO spark.SparkContext: Starting job: showString at
NativeMethodAccessorImpl.java:-2
18/01/03 14:52:47 INFO scheduler.DAGScheduler: Got job 11 (showString at
+------+
|salary|
+------+
|  1000|
|  2000|
|  3000|
|  4000|
|  1000|
```

APACHE SPARK

#to avoid duplicates, we use distinct

```
dfVar.select('name').distinct().show()
18/01/03 14:54:22 INFO scheduler.DAGScheduler: Job 15 finished: showString at
NativeMethodAccessorImpl.java:-2, took 2.197838 s
+-------+
|   name|
+-------+
|Nithesh|
|  Shiva|
|  Dilip|
| Nikhil|
+-------+
```

#crosstab()

```
>>> dfVar.crosstab('name','salary').show()
18/01/03 14:56:39 INFO scheduler.DAGScheduler: Job 17 finished: crosstab at
NativeMethodAccessorImpl.java:-2, took 2.556020 s
+-----------+----+----+----+----+
|name_salary|4000|3000|2000|1000|
+-----------+----+----+----+----+
|      Dilip|   0|   0|   1|   0|
|      Shiva|   0|   1|   0|   0|
|    Nithesh|   1|   0|   0|   0|
|     Nikhil|   0|   0|   0|   2|
+-----------+----+----+----+----+
```

#dropping the duplicates

```
>>> dfVar.select('name').dropDuplicates().show()
-------+
|   name|
+-------+
|Nithesh|
|  Shiva|
|  Dilip|
| Nikhil|
+-------+
```

#dropna() : null data will be dropped

```
>> dfVar.dropna().collect()
18/01/03 15:00:40 INFO spark.SparkContext: Starting job: collect at <stdin>:1
```

APACHE SPARK

18/01/03 15:00:40 INFO scheduler.DAGScheduler: Got job 20 (collect at <stdin>:1) with 1 output partitions

18/01/03 15:00:40 INFO scheduler.DAGScheduler: Job 20 finished: collect at <stdin>:1, took 0.077461 s
[Row(name=u'Nikhil', salary=1000), Row(name=u'Dilip', salary=2000), Row(name=u'Shiva', salary=3000), Row(name=u'Nithesh', salary=4000), Row(name=u'Nikhil', salary=1000)]

#fillna() : a value must be passed . Here we have passed 0. Wherever null values are present, they should be replaced.

```
>>> dfVar.fillna(0).collect()
18/01/03 15:03:08 INFO spark.SparkContext: Starting job: collect at <stdin>:1
18/01/03 15:03:08 INFO scheduler.DAGScheduler: Job 21 finished: collect at <stdin>:1, took 0.094692 s
[Row(name=u'Nikhil', salary=1000), Row(name=u'Dilip', salary=2000), Row(name=u'Shiva', salary=3000), Row(name=u'Nithesh', salary=4000), Row(name=u'Nikhil', salary=1000)]
```

#filter()

```
>>> dfVar.filter(dfVar.salary>2000).collect()
18/01/03 15:06:05 INFO storage.BlockManagerInfo: Removed broadcast_30_piece0 on localhost:49221 in memory (size: 6.3 KB, free: 534.5 MB)
18/01/03 15:06:05 INFO spark.ContextCleaner: Cleaned accumulator 77
18/01/03 15:06:05 INFO storage.BlockManagerInfo: Removed broadcast_29_piece0 on localhost:49221 in memory (size: 6.1 KB, free: 534.5 MB)
18/01/03 15:06:05 INFO scheduler.DAGScheduler: Job 22 finished: collect at <stdin>:1, took 0.047626 s
[Row(name=u'Shiva', salary=3000), Row(name=u'Nithesh', salary=4000)]
```

#count

```
>>> dfVar.groupby('name').count().show()
+-------+-----+
|   name|count|
+-------+-----+
|Nithesh|    1|
|  Shiva|    1|
|  Dilip|    1|
| Nikhil|    2|
+-------+-----+
```

APACHE SPARK

#sample() : It accepts 3 arguments. True-repalcement false- no replacement (eliminates repititions)
Fraction: if we give 0.5 i.e., only 50% of data is only loaded from existing data frame
Next argument is some random sed

```
>>> dfVar.sample(False,0.5,1).show()
18/01/03 15:12:35 INFO spark.SparkContext: Starting job: showString at
NativeMethodAccessorImpl.java:-2
18/01/03 15:12:35 INFO scheduler.DAGScheduler: Got job 25 (showString at
NativeMethodAccessorImpl.java:-2) with 1 output partitions
finished: showString at NativeMethodAccessorImpl.java:-2, took 0.051914 s
+------+------+
|  name|salary|
+------+------+
|Nikhil|  1000|
| Shiva|  3000|
+------+------+

>>> dfVar.sample(False,0.9,1).show()
18/01/03 15:12:54 INFO spark.SparkContext: Starting job: showString at
NativeMethodAccessorImpl.java:-2
18/01/03 15:12:54 INFO scheduler.DAGScheduler: Job 26 finished: showString at
NativeMethodAccessorImpl.java:-2, took 0.062157 s
+-------+------+
|   name|salary|
+-------+------+
| Nikhil|  1000|
|  Shiva|  3000|
|Nithesh|  4000|
+-------+------+
```

Word count use-case

1) First step, we have to split (in this case it is not necessary) and change into key-value pairs

```
>>> dfVar.select('name').map(lambda word: (word,1)).collect()
18/01/03 15:17:05 INFO spark.SparkContext: Starting job: collect at <stdin>:1
18/01/03 15:17:05 INFO scheduler.DAGScheduler: Got job 27 (collect at <stdin>:1) with 1 output
partitions
18/01/03 15:17:05 INFO scheduler.DAGScheduler: Job 27 finished: collect at <stdin>:1, took 0.065928 s
[(Row(name=u'Nikhil'), 1), (Row(name=u'Dilip'), 1), (Row(name=u'Shiva'), 1), (Row(name=u'Nithesh'), 1),
(Row(name=u'Nikhil'), 1)]
```

2) Start counting the each of the values for the unique keys

```
>>> dfVar.select('name').map(lambda word: (word,1)).reduceByKey(lambda v1,v2: v1 + v2).collect()
18/01/03 15:22:27 INFO scheduler.DAGScheduler: Job 29 finished: collect at <stdin>:1, took 0.212423 s
[(Row(name=u'Dilip'), 1), (Row(name=u'Nikhil'), 2), (Row(name=u'Nithesh'), 1), (Row(name=u'Shiva'), 1)]
```

orderBy():

```
>>> dfVar.orderBy(dfVar.salary).collect()

[Row(name=u'Nikhil', salary=1000), Row(name=u'Nikhil', salary=1000), Row(name=u'Dilip', salary=2000), Row(name=u'Shiva', salary=3000), Row(name=u'Nithesh', salary=4000)]
```

Descending order

```
>>> dfVar.orderBy(dfVar.salary.desc()).collect()

[Row(name=u'Nithesh', salary=4000), Row(name=u'Shiva', salary=3000), Row(name=u'Dilip', salary=2000), Row(name=u'Nikhil', salary=1000), Row(name=u'Nikhil', salary=1000)]
```

withColumn() : it accepts 2 arguments. The first one is new column name, and the second one is how you want (source). Here we are doubling the salary by 2times

```
>>> dfVar.withColumn('new_sal', dfVar.salary*2).select('name','salary','new_sal').show()
18/01/03 15:30:36 INFO spark.SparkContext: Starting job: showString at NativeMethodAccessorImpl.java:-2
18/01/03 15:30:36 INFO scheduler.DAGScheduler: Got job 34 (showString at NativeMethodAccessorImpl.java:-2) with 1 output partitions
18/01/03 15:30:36 INFO scheduler.DAGScheduler: Job 34 finished: showString at NativeMethodAccessorImpl.java:-2, took 0.069361 s
+-------+------+-------+
|   name|salary|new_sal|
+-------+------+-------+
| Nikhil|  1000|   2000|
|  Dilip|  2000|   4000|
|  Shiva|  3000|   6000|
|Nithesh|  4000|   8000|
| Nikhil|  1000|   2000|
+-------+------+-------+
```

APACHE SPARK

Using SCALA

```
[cloudera@quickstart ~]$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
18/01/03 15:37:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
18/01/03 15:37:58 INFO spark.SecurityManager: Changing view acls to: cloudera
18/01/03 15:37:58 INFO spark.SecurityManager: Changing modify acls to: cloudera
18/01/03 15:37:58 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls
disabled; users with view permissions: Set(cloudera); users with modify permissions: Set(cloudera)
18/01/03 15:37:58 INFO spark.HttpServer: Starting HTTP Server
18/01/03 15:37:59 INFO server.Server: jetty-8.y.z-SNAPSHOT
18/01/03 15:37:59 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0:42951
18/01/03 15:37:59 INFO util.Utils: Successfully started service 'HTTP class server' on port 42951.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.5.0-cdh5.5.0
      /_/

18/01/03 15:38:17 INFO session.SessionState: No Tez session required at this point.
hive.execution.engine=mr.
18/01/03 15:38:17 INFO repl.SparkILoop: Created sql context (with Hive support)..
SQL context available as sqlContext.

scala> val lines = sc.parallelize(List("one one one two", "three four four"))
lines: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:21

scala> lines.collect()
18/01/03 15:42:46 INFO spark.SparkContext: Starting job: collect at <console>:24
18/01/03 15:42:46 INFO scheduler.DAGScheduler: Got job 0 (collect at <console>:24) with 1 output
partitions
18/01/03 15:42:46 INFO scheduler.DAGScheduler: Final stage: ResultStage 0(collect at <console>:24)
18/01/03 15:42:46 INFO scheduler.DAGScheduler: Parents of final stage: List()
18/01/03 15:42:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all
completed, from pool
18/01/03 15:42:47 INFO scheduler.DAGScheduler: Job 0 finished: collect at <console>:24, took 0.507757
s
res0: Array[String] = Array(one one one two, three four four)
```

APACHE SPARK

here we have to split the input. Here lambda is declared as =>

```
scala> var words = lines.flatMap(line => (line.split(" ")))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at flatMap at <console>:23

scala> words.collect()
18/01/03 15:46:19 INFO spark.SparkContext: Starting job: collect at <console>:26
18/01/03 15:46:19 INFO scheduler.DAGScheduler: Got job 1 (collect at <console>:26) with 1 output
partitions
18/01/03 15:46:19 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 1)
18/01/03 15:46:19 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1). 961 bytes result sent
to driver
18/01/03 15:46:19 INFO scheduler.DAGScheduler: ResultStage 1 (collect at <console>:26) finished in
0.026 s
18/01/03 15:46:19 INFO scheduler.DAGScheduler: Job 1 finished: collect at <console>:26, took 0.049930
s
res1: Array[String] = Array(one, one, one, two, three, four, four)


scala>  words.map(word => (word,1)).collect()
18/01/03 15:48:09 INFO spark.SparkContext: Starting job: collect at <console>:26
18/01/03 15:48:09 INFO scheduler.DAGScheduler: Got job 2 (collect at <console>:26) with 1 output
partitions
18/01/03 15:48:09 INFO scheduler.DAGScheduler: Final stage: ResultStage 2(collect at <console>:26)
18/01/03 15:48:09 INFO scheduler.DAGScheduler: Job 2 finished: collect at <console>:26, took 0.046487
s
res2: Array[(String, Int)] = Array((one,1), (one,1), (one,1), (two,1), (three,1), (four,1), (four,1))


scala>  words.map(word => (word,1)).reduceByKey(_ + _).collect()
18/01/03 15:49:39 INFO storage.BlockManagerInfo: Removed broadcast_2_piece0 on localhost:53872
in memory (size: 1266.0 B, free: 534.5 MB)
18/01/03 15:49:39 INFO spark.SparkContext: Starting job: collect at <console>:26
18/01/03 15:49:39 INFO scheduler.DAGScheduler: Registering RDD 3 (map at <console>:26)
18/01/03 15:49:39 INFO scheduler.DAGScheduler: Got job 3 (collect at <console>:26) with 1 output
partitions
18/01/03 15:49:40 INFO scheduler.DAGScheduler: Job 3 finished: collect at <console>:26, took 0.255159
s
res3: Array[(String, Int)] = Array((two,1), (one,3), (three,1), (four,2))
```

or we can use reduceByKey as in using python

APACHE SPARK

Always turn off Safemode

hdfs dfsadmin -safemode leave