What is Spark ?

In-memory cluster computing framework for large-scale data processing

Some facts about Apache Spark

- Started in 2009 by AMP Lab at UC Berkeley.
- Graduated from Apache Incubator earlier this year.
- Close to 300 contributors on Github.
- Developed using Scala, with Java and Python APIs.
- Can sit on an existing Hadoop cluster.
- Processes data up to 100x faster than Hadoop Map-Reduce in memory or up to 10x faster in disk.

Who are using Spark?

Alibaba

Amazon

Autodesk

Baidu

Conviva

Databricks

eBay Inc.

Guavus

IBM Almaden

NASA JPL

Nokia S&N

Ooyala

Rocketfuel

Shazam

Shopify

Stratio

Yahoo!

Yandex

Full list at: `https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark`

Few misconceptions around Spark

Misconception #1

You need to know **Scala** or **Java** to use Spark

Misconception #1

You need to know **Scala** or **Java** to use Spark

FALSE

Misconception #2

There are not enough documentations or example codes available to get started on **PySpark**

Misconception #2

There are not enough documentations or example codes available to get started on **PySpark**

FALSE

Misconception #3

Not all Spark features are available for **Python** or **PySpark**

Misconception #3

Not all Spark features are available for **Python** or **PySpark**

FALSE*

Misconception #3

Not all Spark features are available for **Python** or **PySpark**

FALSE*

*** Spark Streaming coming soon!**

PySpark

PySpark

About PySpark

- Python API for Spark using Py4j.
- Provides interactive shell for processing data from command line.
- 2x to 10x less code than standalone programs.
- Can be used from iPython shell or notebook.
- Full support for Spark SQL (previously Shark).
- Spark Streaming coming soon… (version 1.2.0)

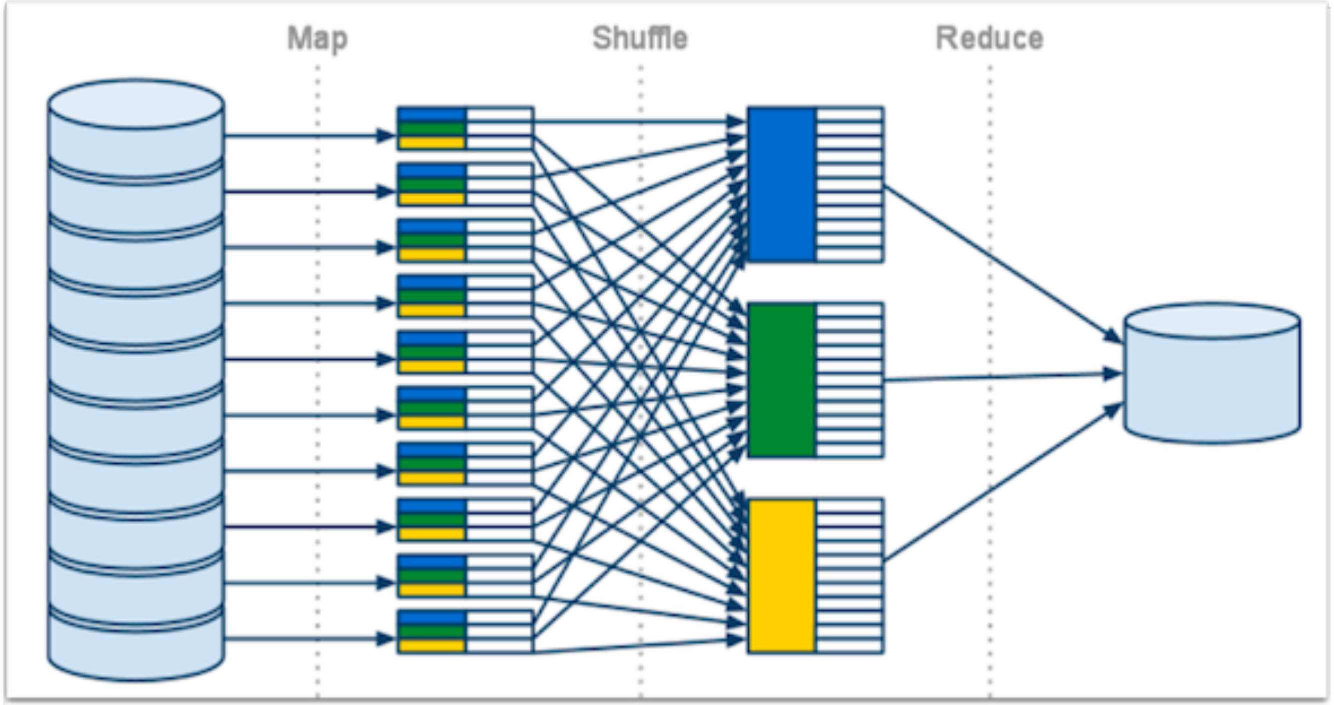
Who can benefit from PySpark?

Data Scientists

- Rich, scalable machine learning libraries (MLlib)
  - Statistics - Correlation, sampling, hypothesis testing
  - ML - Classification, Regression, Collaborative filtering, Clustering, Dimensionality Reduction etc.
- Seamless integration of Numpy, Matplotlib and Pandas for data wrangling and visualizations.
- Advantage of in-memory processing for iterative tasks

Spark vs. Hadoop Map-Reduce

Hadoop Map-Reduce



- A programming paradigm for batch processing.
- Data loaded and read from disk for each iteration and finally written to disk.
- Fault tolerance achieved through data replication on data nodes.
- Each Pig/Hive query spawns a separate Map-Reduce job and reads from disk.

What is different in Spark?

- Data is cached in RAM from disk for iterative processing.
- If data is too large for memory, rest is spilled into disk.
- Interactive processing of datasets without having to reload in the memory.
- Dataset is represented as RDD (Resilient Distributed Dataset) when loaded into Spark Context.
- Fault tolerance achieved through RDD and lineage graphs.

RDD (Resilient Distributed Dataset)

What is RDD?

- A read-only collection of objects, partitioned across a set of machines.

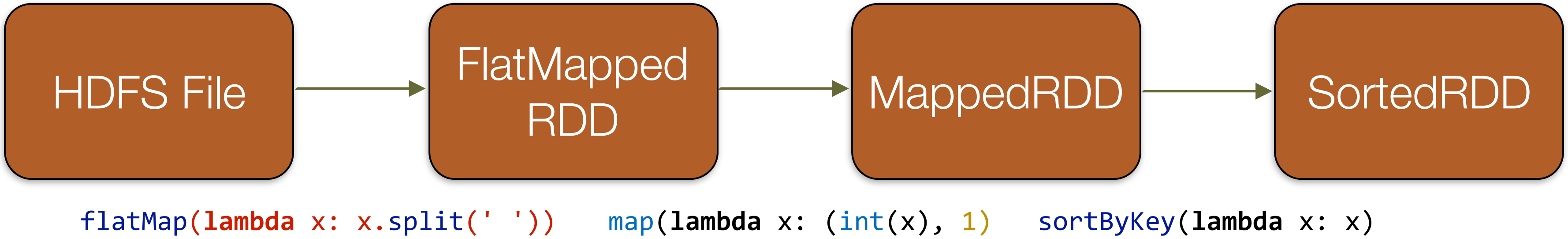
- RDDs can be re-built if a partition is lost through **lineage**: an RDD has information about how it was derived from other RDDs to be reconstructed.

- RDDs can be cached and reused in multiple Map-Reduce like parallel operations.

- RDDs are lazy and ephemeral.

RDD Lineage

```
lines = sc.textFile("hdfs://...")
sortedCount = lines.flatMap(lambda x: x.split(' ')) \
                   .map(lambda x: (int(x), 1)) \
                   .sortByKey(lambda x: x)
```



RDD Operations

Transformations	map, filter, flatmap, sort
Actions	reduce, count, collect, save

Map

Returns a new RDD by applying a function to each element of this RDD

```
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'map_example')

rdd = sc.parallelize(["banana", "apple",
“watermelon"])
sorted(rdd.map(lambda x: (x, len(x))).collect())

[('apple', 5), ('banana', 6), ('watermelon',
10)]
```

FlatMap

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

```
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'flatmap_example')

rdd = sc.parallelize(["this is you", "you are here",
"how do you feel about this"])
sorted(rdd.flatMap(lambda x: x.split()).collect())

['about', 'are', 'do', 'feel', 'here', 'how', 'is',
'this', 'this', 'you', 'you', 'you']
```

Filter

Returns a new RDD containing only the elements that satisfy a predicate.

```
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'filter_example')

rdd = sc.parallelize([1, 2, 3, 4, 5])
rdd.filter(lambda x: x % 2 == 0).collect()
```

```
[2, 4]
```

Reduce

Reduces the elements of this RDD using the specified commutative and associative binary operator. Currently reduces partitions locally.

```python
from operator import add
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'reduce_example')

num_list = [num for num in xrange(1000000)]
sc.parallelize(num_list).reduce(add)
```

```
499999500000
```

Count

Return the number of elements in this RDD.

```
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'count_example')

file = sc.textFile("hdfs://...")
file.flatMap(lambda line: line.split()).count()
```

```
4929075
```

SaveAsTextFile

Save this RDD as a text file, using string representations of elements.

```
from pyspark.context import SparkContext

sc = SparkContext('local[2]', 'filter_example')

file = sc.textFile(“hdfs://...")

file.flatMap(lambda line: line.split())
    .saveAsTextFile(“output_dir”)
```