Algorithms and Applications of Data Mining - Introduction to Spark

Yijun Lin

Thanks for source slide and material to: Dr. Heather Miller
https://www.coursera.org/learn/scala-spark-big-data/home/welcome

Why Need MapReduce

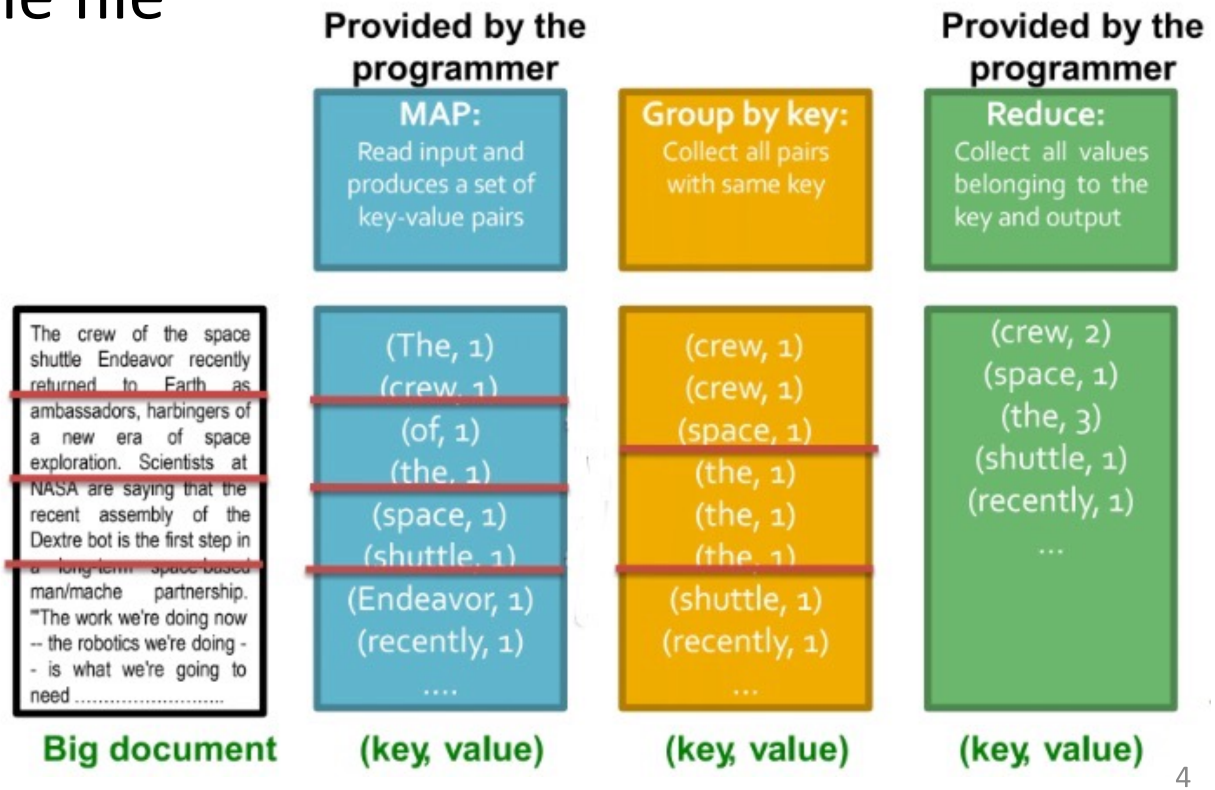
- Challenges in large-scale computing
  - How to distribute computation (moving data around is time-consuming)
  - How to make it easy to write distributed programs
  - How to handle machine failures

What Is MapReduce

- MapReduce addresses these problems
  - Master Node: Coordinate worker nodes
  - Map worker/node: Extract something you care about, usually data are represented as (key, value) pairs
  - Reduce worker/node: Aggregate, summarize, ...

MapReduce: Word Count

- Task: We have a huge text document, and we want to count the number of times each distinct word appears in the file



Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "'The work we're doing now -- the robotics we're doing -- is what we're going to need ....

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

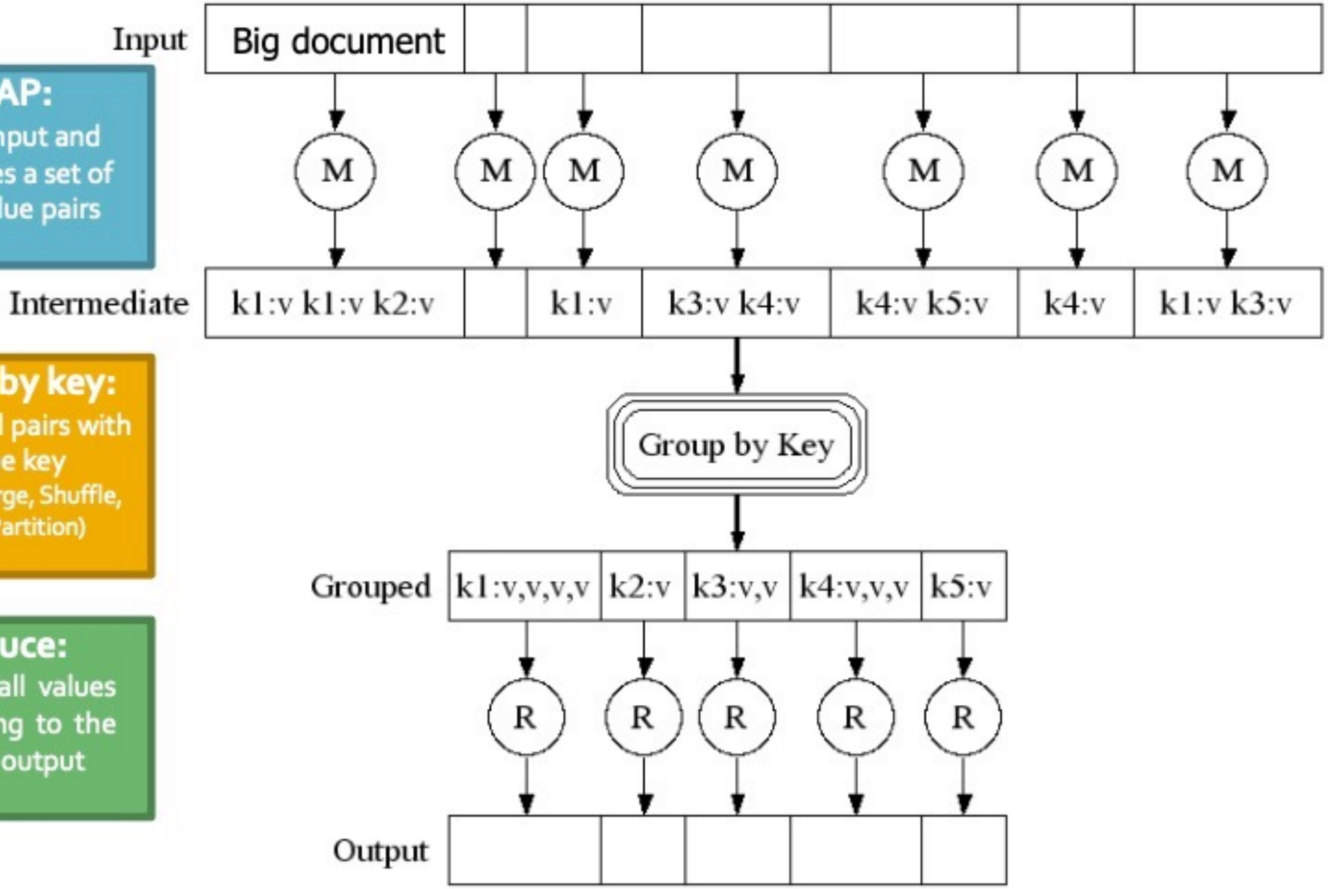
(key, value)

MapReduce: The Diagram

MAP:
Read input and produces a set of key-value pairs

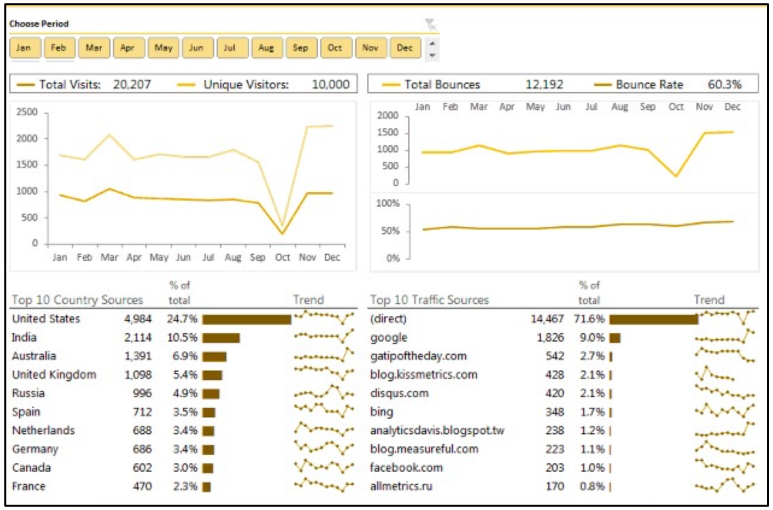
Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



What Is Spark

- **Apache Spark** is an open-source cluster-computing framework for large-scale data processing

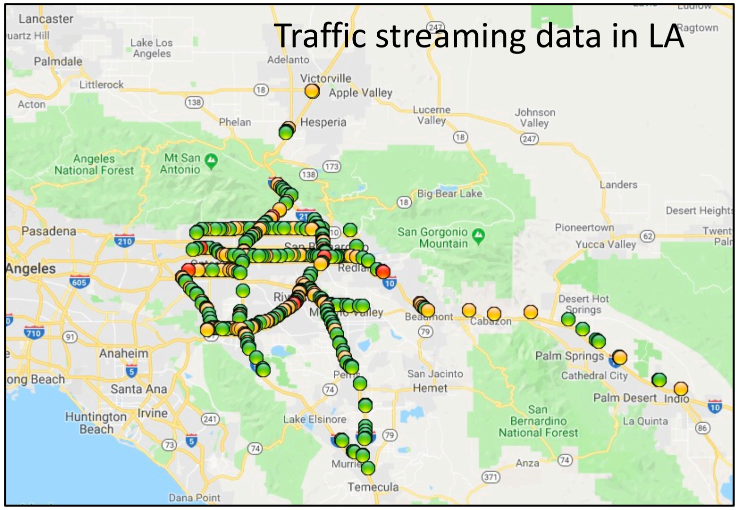


Total Visits: 20,207 | Unique Visitors: 10,000

Total Bounces 12,192 | Bounce Rate 60.3%

Top 10 Country Sources		% of total	Trend
United States	4,984	24.7%	
India	2,114	10.5%	
Australia	1,391	6.9%	
United Kingdom	1,098	5.4%	
Russia	996	4.9%	
Spain	712	3.5%	
Netherlands	688	3.4%	
Germany	686	3.4%	
Canada	602	3.0%	
France	470	2.3%	

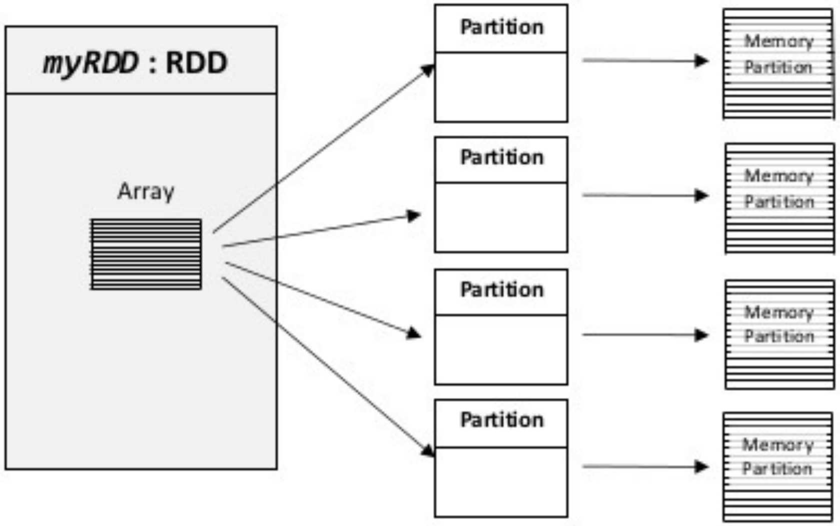
Top 10 Traffic Sources		% of total	Trend
(direct)	14,467	71.6%	
google	1,826	9.0%	
gatipoftheday.com	542	2.7%	
blog.kissmetrics.com	428	2.1%	
disqus.com	420	2.1%	
bing	348	1.7%	
analyticsdavis.blogspot.tw	238	1.2%	
blog.measureful.com	223	1.1%	
facebook.com	203	1.0%	
allmetrics.ru	170	0.8%	


Traffic streaming data in LA

How Does Spark Work?

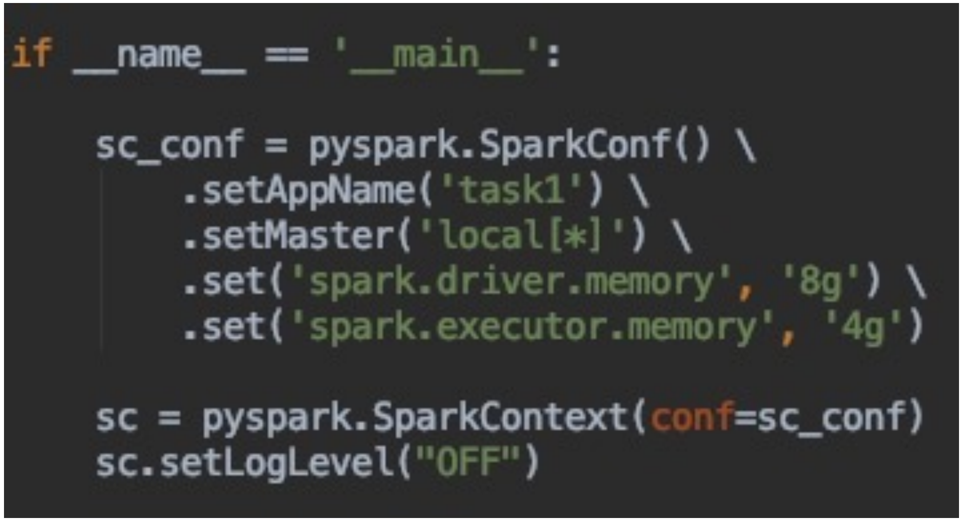
- Resilient Distributed Datasets (RDDs)
- An **immutable**, **in-memory collection** of objects
- Each RDD can be split into multiple partitions, which in turn are computed on different nodes of the cluster

- RDDs are like collections
  - RDD[T] and List[T]



SparkContext Object

- Create a SparkContext Object to start
  - Can be thought as your handle to the Spark Cluster



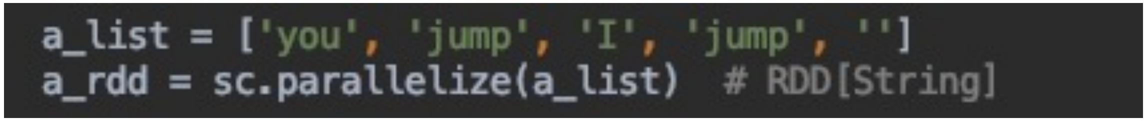
```
if __name__ == '__main__':

    sc_conf = pyspark.SparkConf() \
        .setAppName('task1') \
        .setMaster('local[*]') \
        .set('spark.driver.memory', '8g') \
        .set('spark.executor.memory', '4g')

    sc = pyspark.SparkContext(conf=sc_conf)
    sc.setLogLevel("OFF")
```

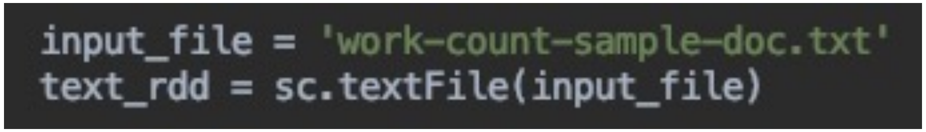
How To Create An RDD

- Create from a SparkContext object
  - **Parallelize:** convert a local collection to an RDD



```
a_list = ['you', 'jump', 'I', 'jump', '']
a_rdd = sc.parallelize(a_list)  # RDD[String]
```

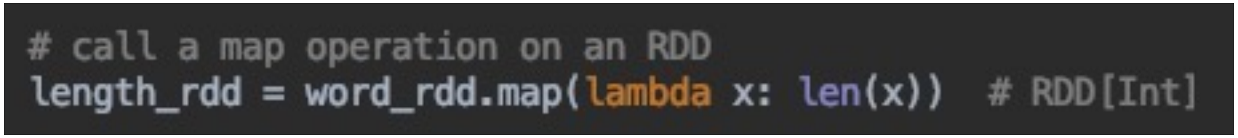
  - **TextFile**: read a file from HDFS or local file system



```
input_file = 'work-count-sample-doc.txt'
text_rdd = sc.textFile(input_file)
```

How To Create An RDD (Cont.)

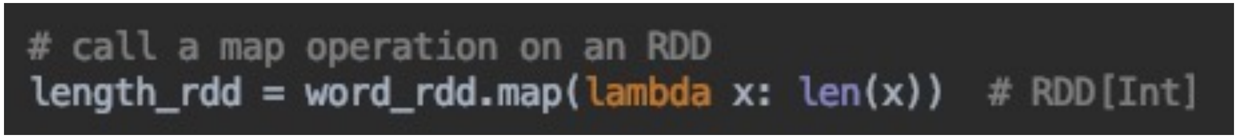
- Transform from an existing RDD
  - E.g., calling a *map operation* on an existing RDD, it will return a new RDD



```
# call a map operation on an RDD
length_rdd = word_rdd.map(lambda x: len(x))  # RDD[Int]
```

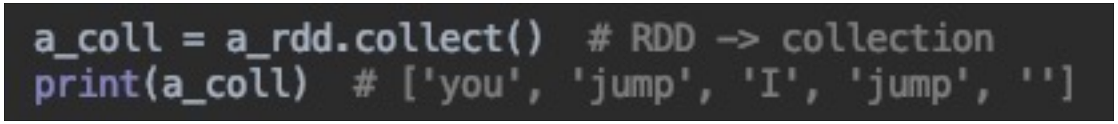
RDD Operations

- Transformations:
  - E.g., map, filter, ...



```
# call a map operation on an RDD
length_rdd = word_rdd.map(lambda x: len(x))  # RDD[Int]
```

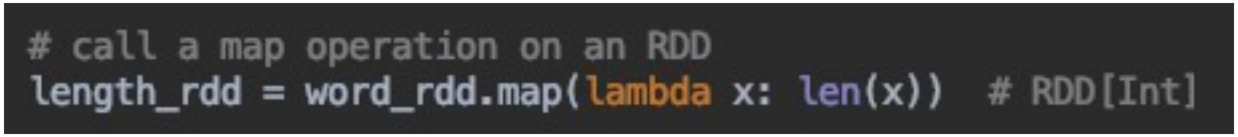
- Actions:
  - E.g., collect, reduce, ...



```
a_coll = a_rdd.collect()  # RDD -> collection
print(a_coll)  # ['you', 'jump', 'I', 'jump', '']
```

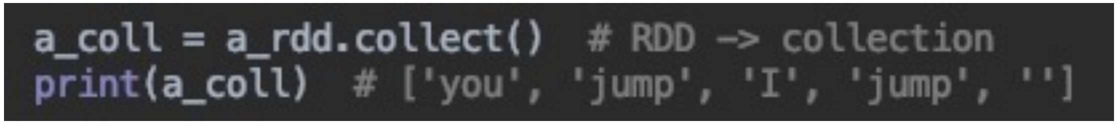
Transformations VS. Actions

- Transformations
  - Return new RDDs as results
  - They are **lazy**, the result RDD is **not immediately** computed



```
# call a map operation on an RDD
length_rdd = word_rdd.map(lambda x: len(x))  # RDD[Int]
```

- Actions
  - Compute a result based on an RDD, and returned
  - They are **eager**, the result is **immediately computed**



```
a_coll = a_rdd.collect()  # RDD -> collection
print(a_coll)  # ['you', 'jump', 'I', 'jump', '']
```

Common Transformations

map

map[T](f: A=>B): RDD[T]
Apply function to each element in the RDD and return an RDD of the result.

flatmap

flatmap[T](f: A=>B): RDD[T]
Apply function to each element in the RDD and return an RDD of the result, but output is flattened.

filter

filter[T](pred: A=>Boolean): RDD[T]
Apply predicate function, pred, to each element in the RDD and return an RDD of elements that passed the condition.

distinct

distinct():RDD[T]
Return an RDD with duplicates removed

Common Actions

collect **collect: Array[T]**
Return all elements from RDD.

count **count(): Long**
Return the number of elements in the RDD.

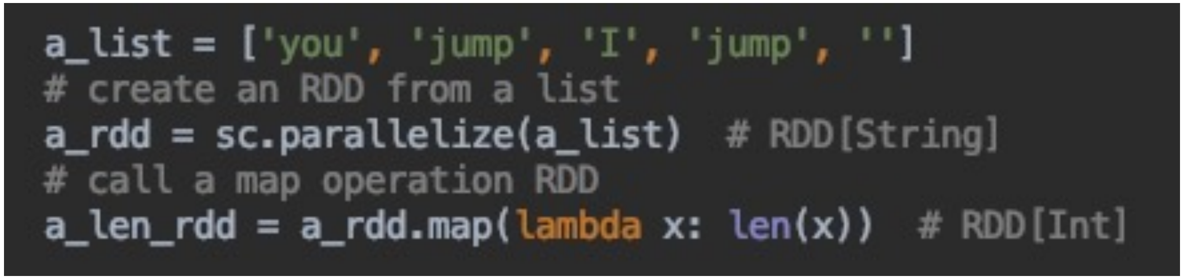
take **take(num: Int): Array[T]**
Return the first <num> elements of the RDD.

reduce **reduce(op: (A, A) => A): A**
Combine the elements in the RDD together using op function and return result.

foreach **foreach(f: A => Unit): Unit**
Apply function to each element in the RDD and return Unit.

Example

- Consider the following example:

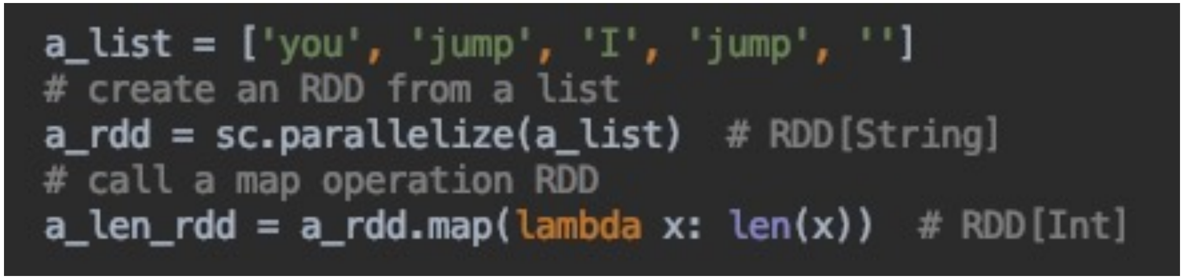


```
a_list = ['you', 'jump', 'I', 'jump', '']
# create an RDD from a list
a_rdd = sc.parallelize(a_list)  # RDD[String]
# call a map operation RDD
a_len_rdd = a_rdd.map(lambda x: len(x))  # RDD[Int]
```

What has happened on the cluster at this point?

Example (Cont.)

- Consider the following example:



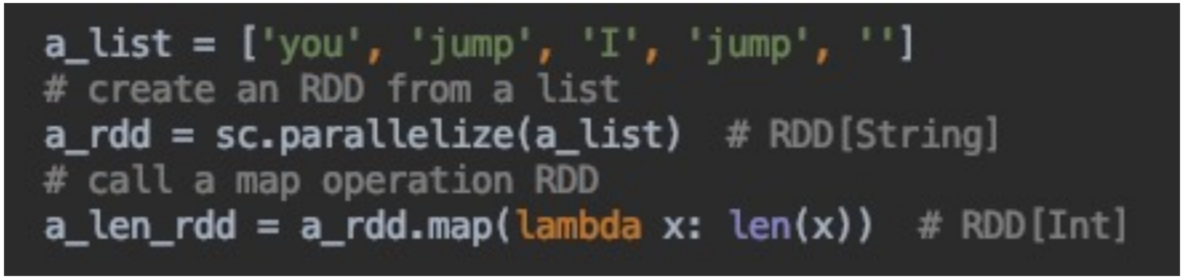
```
a_list = ['you', 'jump', 'I', 'jump', '']
# create an RDD from a list
a_rdd = sc.parallelize(a_list)  # RDD[String]
# call a map operation RDD
a_len_rdd = a_rdd.map(lambda x: len(x))  # RDD[Int]
```

What has happened on the cluster at this point?

Nothing. Execution of map (a transformation) is deferred.

Example (Cont.)

- Consider the following example:

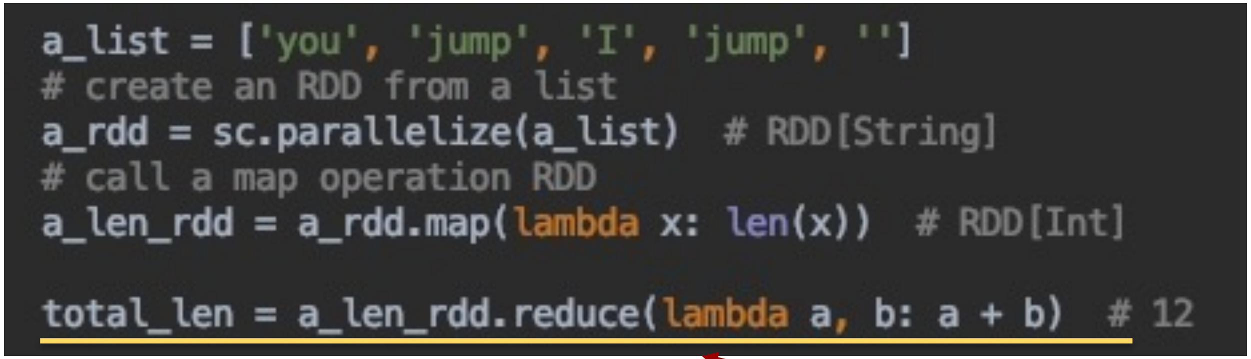


```
a_list = ['you', 'jump', 'I', 'jump', '']
# create an RDD from a list
a_rdd = sc.parallelize(a_list)  # RDD[String]
# call a map operation RDD
a_len_rdd = a_rdd.map(lambda x: len(x))  # RDD[Int]
```

How to ensure this computation is done on the cluster?

Example (Cont.)

- Consider the following example:



```
a_list = ['you', 'jump', 'I', 'jump', '']
# create an RDD from a list
a_rdd = sc.parallelize(a_list)  # RDD[String]
# call a map operation RDD
a_len_rdd = a_rdd.map(lambda x: len(x))  # RDD[Int]

total_len = a_len_rdd.reduce(lambda a, b: a + b)  # 12
```

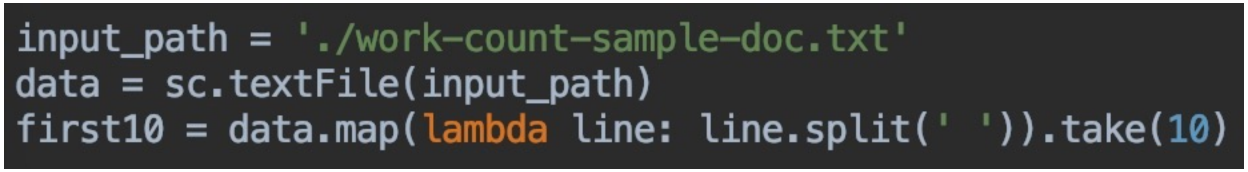
add an action, *reduce*

Spark starts the execution when an action is called

Return the total number of characters in the entire RDD of strings

Benefits of Laziness

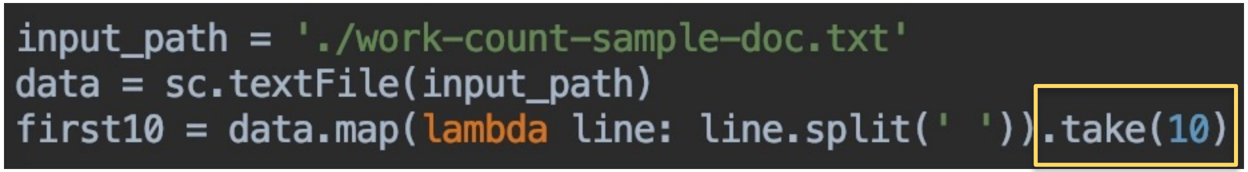
- Another example:



```
input_path = './work-count-sample-doc.txt'
data = sc.textFile(input_path)
first10 = data.map(lambda line: line.split(' ')).take(10)
```

Benefits of Laziness (Cont.)

- Another example:

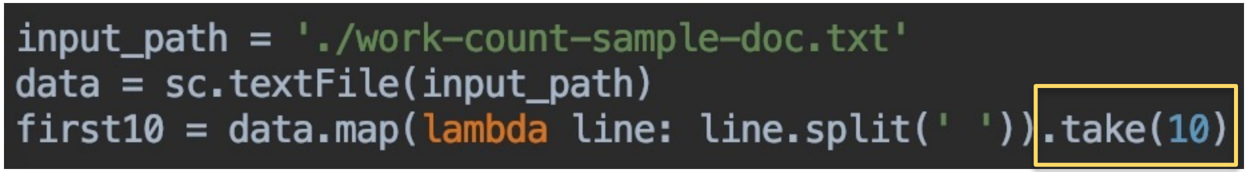


```
input_path = './work-count-sample-doc.txt'
data = sc.textFile(input_path)
first10 = data.map(lambda line: line.split(' ')).take(10)
```

  - The execution of filter is deferred until the take happens
  - Spark will not compute intermediate RDDs. As soon as 10 elements are picked, first10 is done.

Benefits of Laziness (Cont.)

- Another example:



```
input_path = './work-count-sample-doc.txt'
data = sc.textFile(input_path)
first10 = data.map(lambda line: line.split(' ')).take(10)
```

- Spark leverages this by analyzing and optimizing the chain of operations before executing it
- Spark saves time and space to compute elements of the unused result of the filter operation

Word Count Example

- Let’s start!

If You Want To Learn More

- Official documentation
  - http://spark.apache.org/docs/latest/

- Coursera: Big Data Analysis with Scala and Spark

- Books
  - Learning Spark, O’ Reilly
  - Advanced Analytics with Spark: Patterns for Learning from Data at Scale, O’ Reilly
  - Machine Learning with Spark, Packt