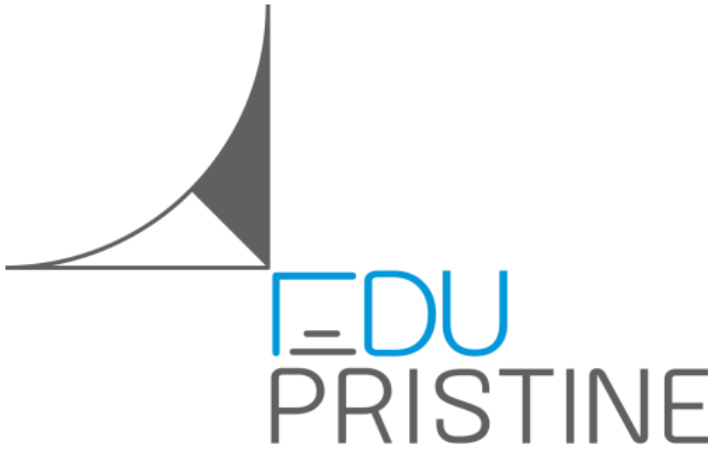


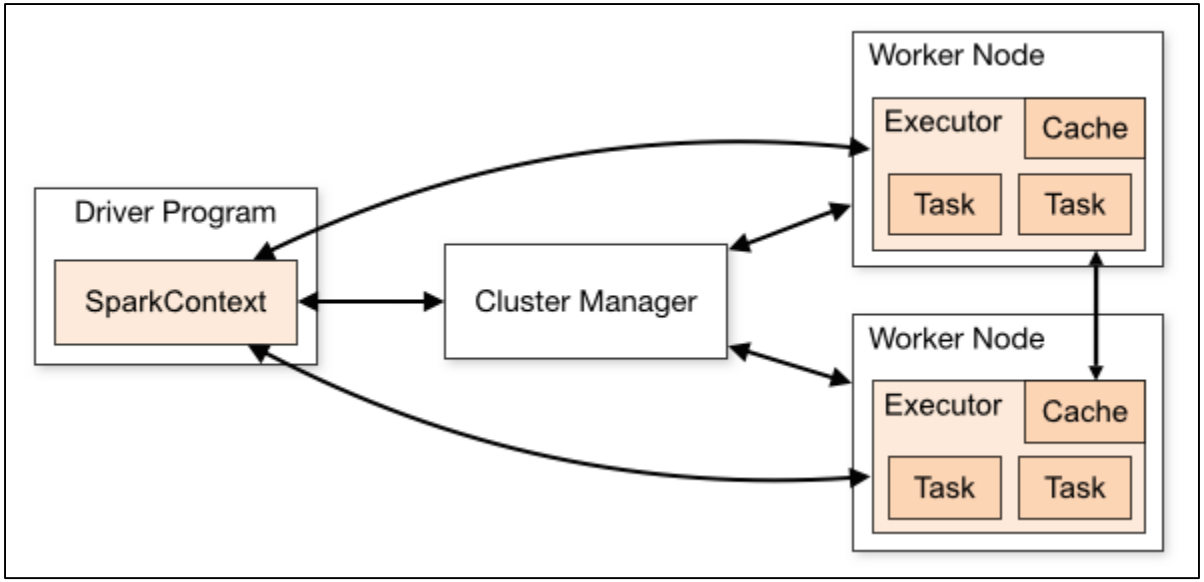
Spark Lab

Agenda

- Spark Architecture
- Spark Context
- Resilient Distributed Dataset
- Intro to Python
- Running pyspark:
  - Interactive
  - Batch
- Transformations
- Actions
- Caching RDD's
- Spark Program Life Cycle
- Key Value Transformations
- Word count
- Closures, Accumulators and Broadcast variables
- Batch processing with Spark

Spark Drivers and workers

- Spark has two main programs, driver and worker program.
- Spark context object in your driver program, will coordinate with cluster manager. In hadoop it will he resource manager.
- Get the required resources for running the job.
- Now connects to executors in worker node, all the computations run in worker nodes where data is residing. In hadoop, it would be data nodes.



Spark Architecure; key points

- Each application gets its own executor processes.
- Spark is agnostic to the underlying cluster manager. It can work with Mesos, YARN.
- Driver will be coordinating with workers.
- Spark supports three cluster managers:
  - Standalone
  - Apache Mesos
  - Hadoop YARN
- In our lab, Spark will be coordinating with **Yet Another Resource Negotiator(YARN)**

Spark Context

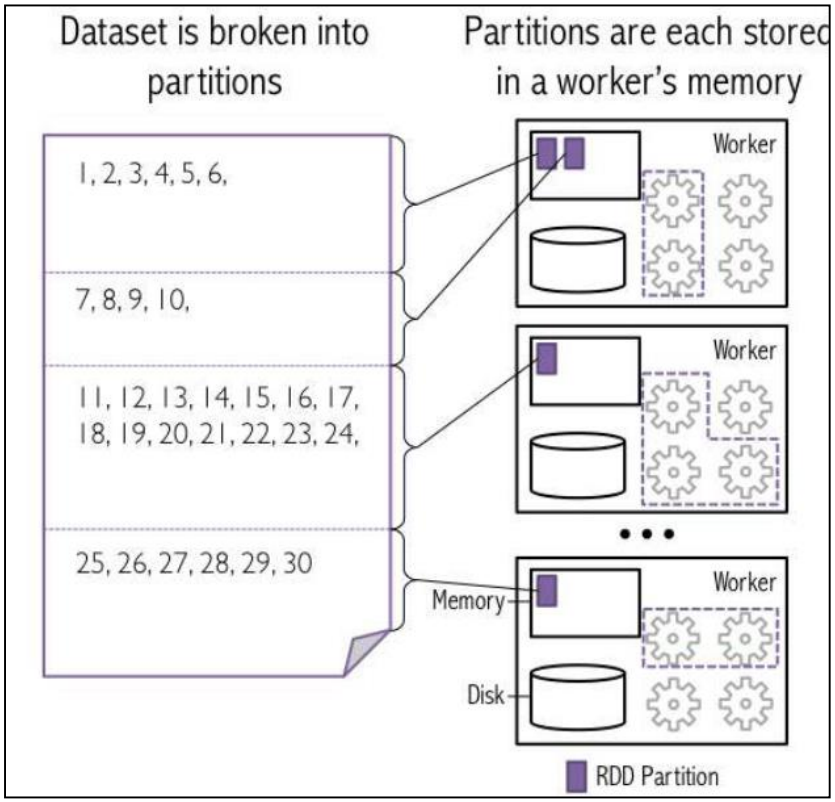
- Driver program in spark creates spark context object.
- Configurations in spark context will decide in which mode it will be running. If spark context is configured with YARN, it will run on Hadoop.
- In interactive analysis, spark context will be automatically created as sc.
- In batch programming it should be explicitly created by using SparkContext class.

Resilient distributed dataset

- Primary abstraction in spark:
  - Fault tolerant collection of elements which can be operated parallel
  - Immutable once constructed
  - Track lineage information to efficiently compute lost data
- You construct RDDs:
  - parallelizing an existing collection in your driver program
  - referencing a dataset in an external storage system such as HDFS, Hbase, S3
  - Or from existing transformations

RDDs

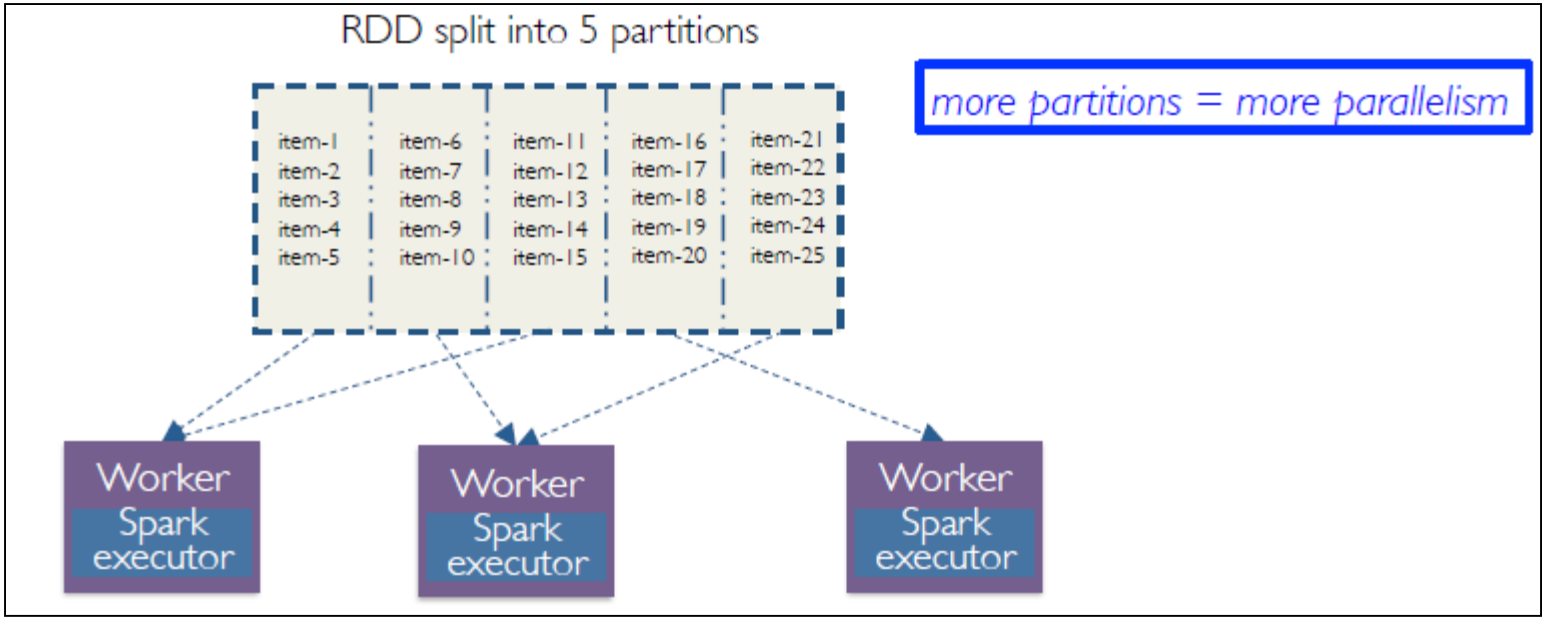
- Programmer specifies the number of partitions for an RDD.



Then partitions are stored in workers memory. Each partition can be allocated a core on the Worker machine, when the actual execution begins

More partitions, more parallelism

- Default would be chosen, if not specified.
- If spark is running on Hadoop, user need not mention the number of partitions, as data is already partitioned on HDFS.

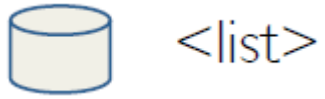


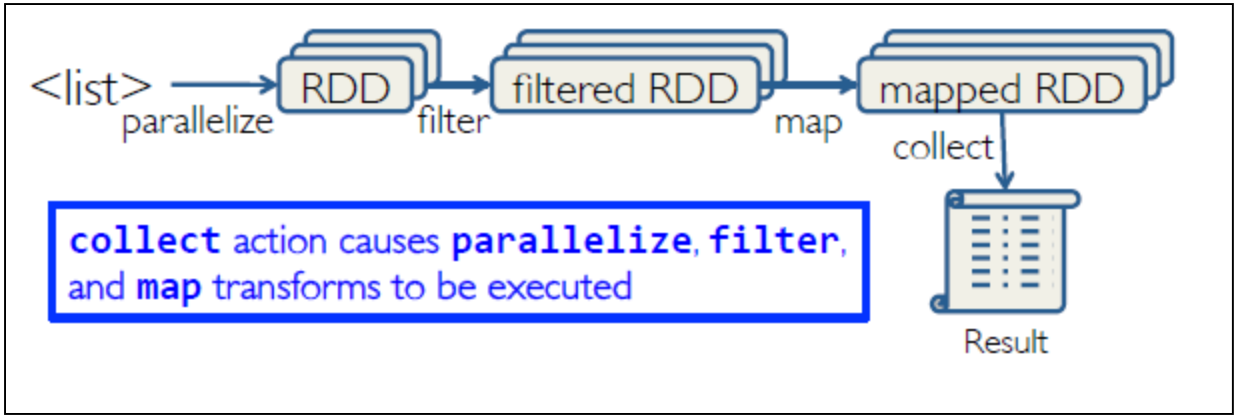
RDD Operations

- Two types of operations can be performed on RDD's, Transformations and actions.
- Transformations are lazy
- In-fact Transformations will be only executed when action runs on it.

Working with RDD's

- Create an RDD from data source:
- Apply transformations
- Apply actions to get results out of it.





- In this example, parallelize is a function to create RDD from list.
- Filter and Map are transformations.
- Collect is the action, which starts the entire process.

Creating RDD's

- From existing collections in Python:
  - list = [1,2,3,4,5]
  - rdd = sc.parallelize(list,4)

- Here 4 refers to number of partitions.

- From raw files in HDFS or any other external system.

rdd = sc.textFile('PathOfFile',4)

Spark Transformations

- Creates new datasets from existing one's.
- All spark transformations are lazy
- Using lazy evaluation, spark does not compute results right away. Spark remembers set of transformations applied to base dataset
  - Spark optimizes the required calculations
  - Spark recovers from failures and slow workers
- Think of this a recipe for creating end result.

Review Python lambda functions

- Small anonymous functions(not bound to any name)

  lambda a,b: a+b

  a,b are inputs

  a+b is what it returns

- Restricted to a single expression
- Can be used where ever function objects are required.

Key notes before we start the lab

- Spark is more similar to functional programming language, so you as developer, get to know the functions available in spark, use them wisely to solve your data analysis problems.

- Like map reduce, don't try to focus on writing complex programs, keep it simple, get your logic write, and then start programming.

- Reason for choosing python for this lab, its simple, easy and fun to learn. Python has rich set of libraries for data analys

- If you know spark programming in python, you can also do similar stuff in Scala. Since spark is functional programming language, functions will be similar across different programming platforms, syntax and semantics might change!

Spark is both interactive and batch. Since there are many users accessing lab environment in parallel, we chose to use batch processing.

Instructors will demonstrate you interactive shell.

Transformations

- map
- filter
- distinct
- union
- Intersection
- flatMap

map

Description

- Returns a new distributed dataset formed by passing each element of a source through a function func.
- In spark, map function works on individual elements, and can also be used to create key value pair RDD. Element level operations like split, concatenate, lower, upper can be used with map.
- In SQL it can be compared to select operation.

Code:

```
### Create source RDD, the same will be used for the coming examples
sourceRDD = sc.parallelize([51,44,11,12,14,5,6,3,3,1,7,7,11],4)

### Transform elements using map
mapRdd = sourceRDD.map(lambda x:x+1)

### Get the results out of RDD using action collect
mapResult = mapRdd.collect()

### Print the results
print(mapResult)
```

Result

```
[52, 45, 12, 13, 15, 6, 7, 4, 4, 2, 8, 8, 12]
```

filter

Description:

Return a new dataset formed by selecting those elements of the source on which func returns true.

Examples: Customers from a particular region, products with sale more than 1000.

In SQL it can be compared to where condition.

Code :

```
# Filtering condition to get even elements
filterRDD = sourceRDD.filter(lambda x:x%2==0)

# Get the result from RDD using collect
action
filterResult = filterRDD.collect()

# check the output
print(filterResult)
```

Result:

```
[44, 12, 14, 6]
```

Distinct

Description :

Return a new dataset that contains the distinct elements of the source dataset.

Example: Distinct number of customers who have done transactions today.

In SQL, it can be compared to distinct.

Code :

```
# Distinct function to get distinct elements
distinctRDD = sourceRDD.distinct()

# Get the results from distinctRDD
distinctResult = distinctRDD.collect()

# Print results
print(distinctResult)
```

Result :

```
[1, 3, 5, 6, 7, 12, 11, 44, 14, 51]
```

```
if you see, sourceRDD has two 11, after distinct you see only one 11.
```

Union

Description :

Returns union of two datasets.

Similar to union function in SQL.

Code :

```
# Create two sample RDD's
rdd1 = sc.parallelize([1,6,12,5,8,9])
rdd2 = sc.parallelize([1,12,11,5,9,7])

# union on Two RDD's
unionRDD = rdd1.union(rdd2)

# Get the result
unionResult = unionRDD.collect()

print(unionResult)
```

Result :

```
[1, 6, 12, 5, 8, 9, 1, 12, 11, 5, 9, 7]
```

Intersection

Description :

In mathematics, the intersection A ∩ B of two sets A and B is the set that contains all elements of A that also belong to B but no other elements.

Code :

```
# Intersection on Two RDD's
intersectionRDD = rdd1.intersection(rdd2)

# Get the result
intersectionResult = intersectionRDD.collect()
print(intersectionResult)
```

Result:

```
[12, 1, 5, 9]
```

FlatMap

Description :

similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).

Code

1

```
# Let us find word count
linesRDD=sc.parallelize(["map reduce is spark","spark is version
of map reduce","Spark map reduce next version"])
```

2

```
# split function in python would return list of words
wordsList=linesRDD.map(lambda line:line.split(" "))

# Check the output
wordsListOp = wordsList.collect()
print(wordsListOp)

[['map', 'reduce', 'is', 'spark'],
['spark', 'is', 'version', 'of', 'map', 'reduce'],
['Spark', 'map', 'reduce', 'next', 'version']]

# Now if we even do a count, what we get is sentences
count, not word count
sentenceCount = wordsList.count() print(sentenceCount)
3
```

3

```
# lets try flat map
words=linesRDD.flatMap(lambda line:line.split(" "))

# check the output
wordsop=words.collect()
print(wordsop)
['map', 'reduce', 'is', 'spark', 'spark', 'is',
'version', 'of', 'map', 'reduce', 'Spark', 'map',
'reduce', 'next', 'version']

# Since the words are seperated, word count is easy to make
wordCount=words.count()
print(wordCount)
15
```

How to get results out of Transformations?

```
lines = sc.textFile("pathoffile")

comments = lines.filter(someLogic)
```

- Spark will not execute as transformations are lazy.
- Spark remembers the transformations, and they act as a recipe.

Spark Actions

- Actions causes spark to execute the transformations.
- Actions are the mechanism to get results out of transformations

Actions

- reduce
- collect
- count
- first
- take
- takeOrdered

reduce

Description:

aggregate dataset's elements using function func. func takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel

Code :

```
# Get the total sum of all number in source
RDD
total = sourceRDD.reduce(lambda a,b:a+b)

# Check the results
print(total)

Output: 175
```

```
# now we can get average
average = total*1.0/count

print(average)

13.4615384615
```

count

Description :

Returns the number of elements in RDD

Code:

```
count = sourceRDD.count()

# Check the results
print(count)
13
```

collect

Description:

Return a list that contains all of the elements in this RDD. Note that this method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory.

Code :

```
collectResult = sourceRDD.collect()

# Check the results
print(collectResult)
[51,44,11,12,14,5,6,3,3,1,7,7,11]
```

First

Description:

Returns the first element in RDD

If you want to get an understanding of data, you can use first. This will give you glance of data

Code :

```
first= sourceRDD.first()

# Check the results
print(first)
51
```

take

Description:

Returns the first n elements in RDD. Where n refers to number of elements

Take will be very handy, when you have very large data, and you want to have a look at few records, use take and mention the number of elements

Code :

```
take5= sourceRDD.take(5)

# Check the results
print(take5)
[51,44,11,12,14]
```

Take Ordered

Description :

Return n elements ordered in ascending order or as specified by the optional key function. n refers to number of elements to return.

In SQL it is similar to order function. By default, it only does ascending sort. If you want to get in descending or some other order, then define a function which will help takeOrdered function to get the position of elements to be displayed.

Code :

```
ascending = sourceRDD.takeOrdered(5)

# Check the results
print(ascending)
[1,3,3,5,6]
```

Take Ordered contd..

Description :

Now we need descending order, how to make ascending descending???

Write a function which will negate the elements in RDD. Asceding order of negative values will be descending. This way, although takeOrdered does ascending to find the positions of elements to be displayed in order, it becomes descending

Code :

```
descending = sourceRDD.takeOrdered(5, lambda x:
-1*x)

# Check the results
print(descending)
[54,44,14,12,11]
```

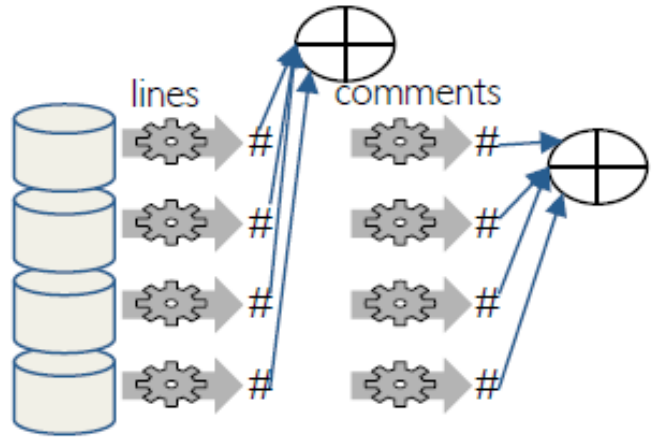
Lets understand spark programming model

- First convert raw data or collections to RDD's.
- Use transformations to transform data.
- Use actions to get results.
- Transformations are lazy.
- Actions will execute transformations and bring out results

But there is a problem…..!!

Consider you are doing multiple computations on the same RDD

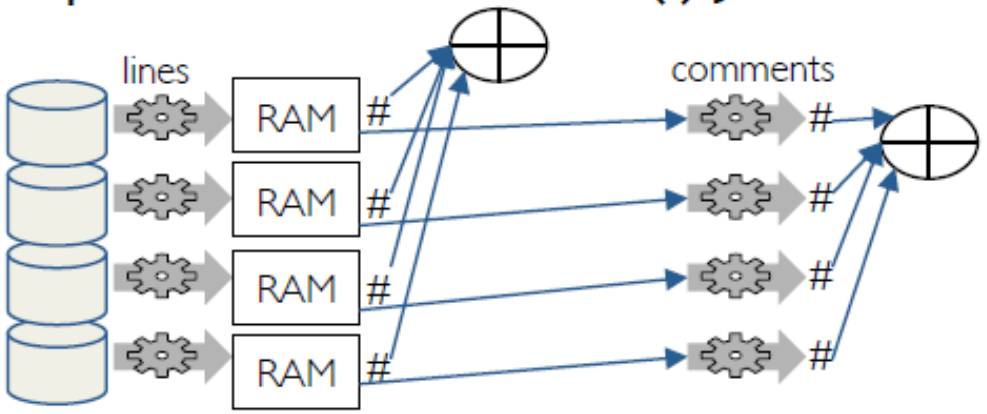
```
lines = sc.textFile("...", 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```



Spark recomputes lines:
- read data (again)
- sum within partitions
- combine sums in driver

Caching RDDs

```
lines = sc.textFile("...", 4)
lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(),comments.count()
```



Spark Program life cycle

- Create RDD's from data on disk or existing collections
- Do relevant transformations based on your data analysis requirements
- Cache frequently used RDD's
- Perform actions to execute parallel computation and produce results

Spark Key Value RDD's

- Like mapreduce, spark supports key-value pairs
- Each element of a pair RDD is pair tuple.
- Key value pairs, only work on pair tuple. By default, the first element in the tuple will become key, and rest will be values.

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])
```

Here a,b,a are keys, 1 is the value

Some key value RDD's

- reduceByKey
- sortByKey
- groupByKey
- join

reduceByKey

Description :

return a new distributed dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type

(V,V) →V

Here function will only work on values, not keys. Don't get confused

Code :

```
# Reduce By Key
keyValueRDD =
sc.parallelize([(1,2),(2,4),(3,2),(1,2),(2,4),(3,2)])
reduceByKeyRDD = keyValueRDD.reduceByKey(lambda a,b:a+b)
results = reduceByKeyRDD.collect()
print(results)
```

sortByKey

Description :

When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.

Syntax = sortByKey([ascending], [numTasks])

If the first argument is true, ascending sort is implemented, false descending.

Code :

```
sortByKeyRDD = keyValueRDD.sortByKey()
sortedResults = sortByKeyRDD.collect()
print(sortedResults)
```

groupByKey

Description

Group by key is like shuffle in map reduce.

All the relevant values of key will be grouped.

After grouping, one can perform a relevant summarization.

groupByKey involves lots of data movement.

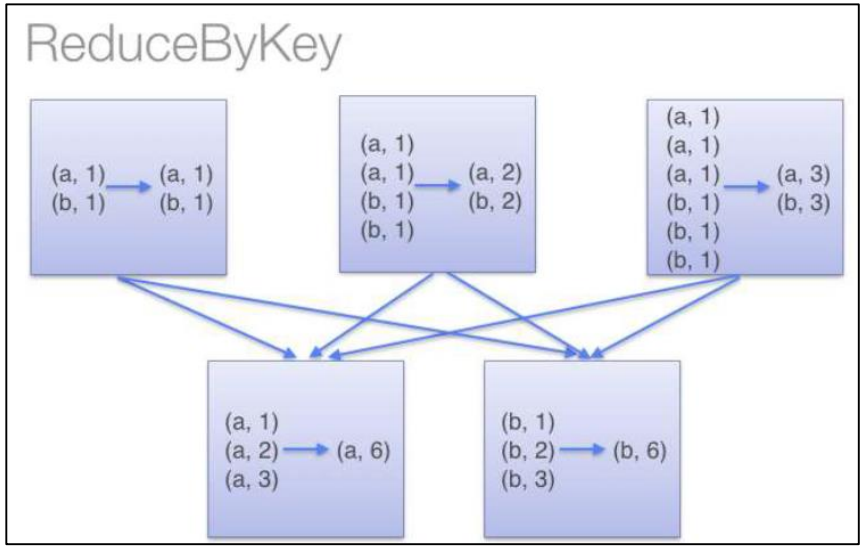
Code :

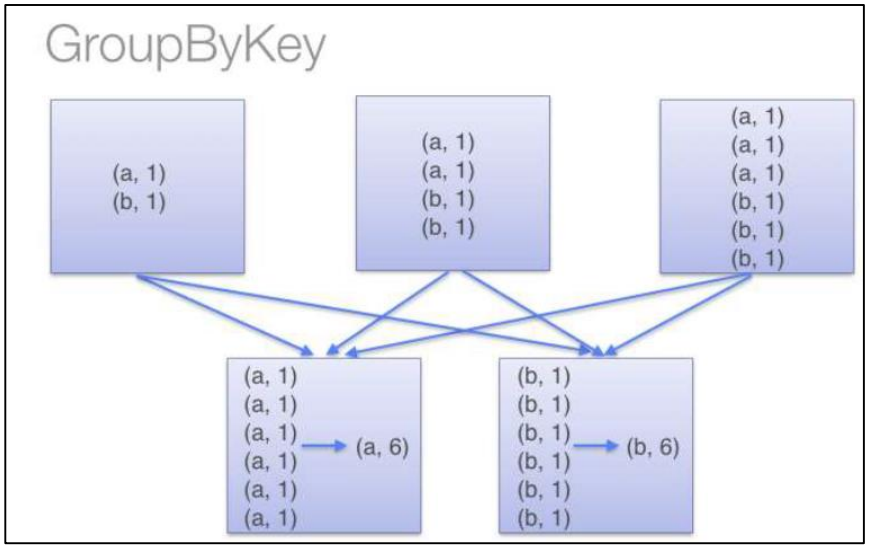
```
groupByKeyRDD = keyValueRDD.groupByKey()
sumResults = groupByKeyRDD.map(lambda (k,v):(k,sum(v))
print(sumResults)
```

groupByKey vs reduceByKey: WordCount Problem

- reduceByKey will do local reduction at each partition, and then will perform global reduction.
- For an overall summary, reduceByKey can be used.
- Example: Region wise sales, Region wise customer sales.

- groupByKey will first shuffle the data, separate each key to different partitions and then do reduction operation.
- In a specific group, if a particular operation needs to be performed, then groupByKey makes sense.
- Example: Each region, top 10 customers.





join

Description :

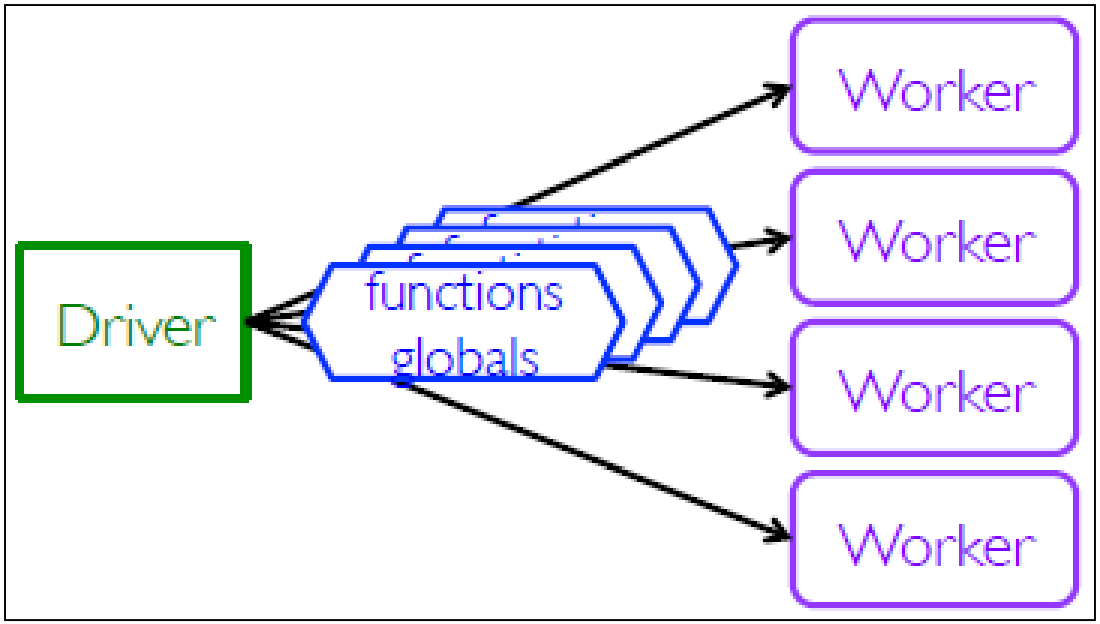
join function, joins two RDD's on the key's.

Code:

```
x = sc.parallelize([("a", 1), ("b", 4)])
y = sc.parallelize([("a", 2), ("a", 3)])
joinedRDD = x.join(y)
joinedResult = joinedRDD.collect()
```

closures

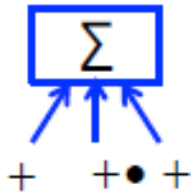
- Spark automatically creates closures for:
  - Functions that run on RDDs at workers
  - Any global variable used by those workers



pySpark Shared Variables

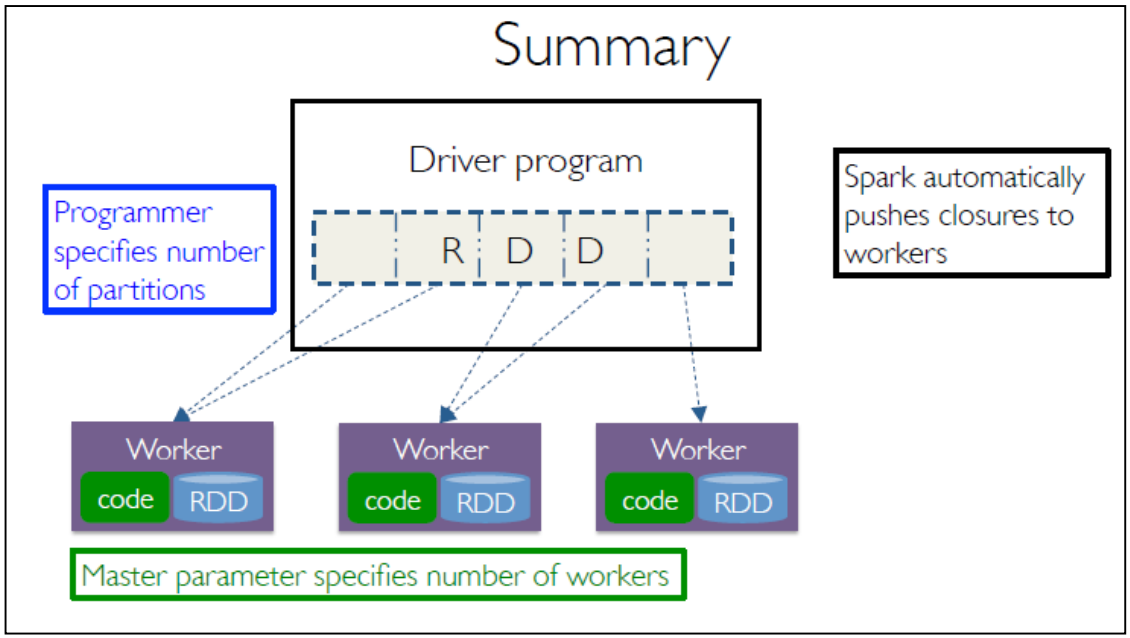
- Broadcast Variables
  - Efficiently send large, read-only value to all workers
  - Saved at workers for use in one or more Spark operations
  - Like sending a large, read-only lookup table to all the nodes
  - Similar to distributed cache in map reduce

- Accumulators
  - Aggregate values from workers back to driver
  - Only driver can access value of accumulator
  - For tasks, accumulators are write-only
  - Use to count errors seen in RDD across workers
  - Similar to counters in map reduce



Summary

- In YARN, driver program will be launched on application master. Workers are containers at node managers.
- In HDFS, you need not mention the number of partitions, as data is already partitioned. If you mention the number of partitions, and it is less than partitions on HDFS, it will not consider. If more, then data partitioned on HDFS will be further partitioned.



Word count on a text file

- Copy UN .txt to HDFS and provide the path of HDFS file

```
fileRDD = sc.textFile("UN.txt")
wordsRDD = fileRDD.flatMap(lambda line:line.split(" "))
wordsRDD.take(4) # Check
wordsKeyValueRDD = wordsRDD.map(lambda word: (word.lower(),1)) #
Convert to lower case
wordsKeyValueRDD.take(4) # Check
wordCountRDD = wordsKeyValueRDD.reduceByKey(lambda a,b:a+b)
wordCountRDD.take(4) # Check
wordCountRDDFlip = wordCountRDD.map(lambda (k,v):(v,k))
wordCountRDDDesc = wordCountRDDFlip.sortByKey(False)
wordCountRDDDesc = wordCountRDDDesc.map(lambda (k,v):(v,k))
wordCountRDDDesc.take(5) # Check # Save the output
wordCountRDDDesc.saveAsTextFile("WordCount")
```

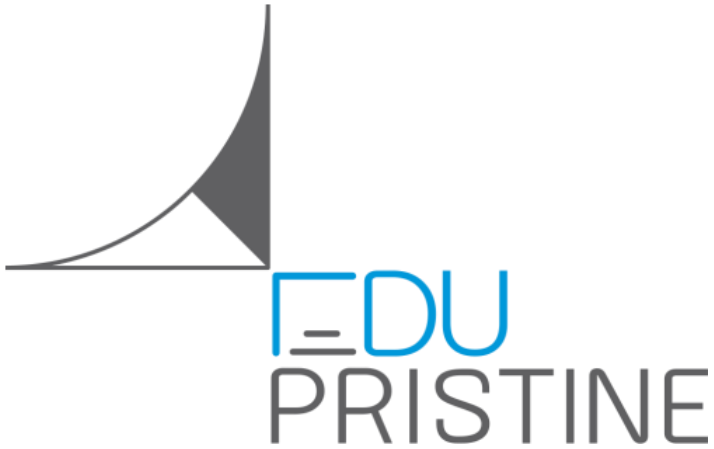
All in one command

```
(fileRDD.flatMap(lambda line:line.split(" "))
        .map(lambda word: (word.lower(),1))
        .reduceByKey(lambda a,b:a+b)
        .map(lambda (k,v):(v,k))
        .sortByKey(False)
        .map(lambda (k,v):(v,k))

.saveAsTextFile("/home/training/Desktop/WordCount2"))
```

Spark batch mode

- Save the commands in a file with .py extension.
- To run it in batch mode, use the following command:
  - spark-submit wordcount.py



Thank You!

help@edupristine.com

www.edupristine.com