

Apache Spark Lab on Docker Cloudera

Contributors: Hareem Qazi - 5371 Tatheer Fatima -09367 Apache Spark is a fast, **in-memory** data processing engine which allows data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets.

Speed

- Run computations in memory.
- Apache Spark has an advanced **DAG** execution engine that supports acyclic data flow and **in-memory computing**.
- **100 times faster** in memory and **10 times faster** even when running on disk than MapReduce.

Generality

- A general programming model that enables developers to write an application by composing arbitrary operators.
- Spark makes it easy to **combine** different processing models seamlessly in the **same application**.
- Example:
 - Data classification through Spark machine learning library.
 - Streaming data through source via Spark Streaming.
 - Querying the resulting data in real time through Spark SQL.

RDD (Resilient Distributed Datasets)

What is a dataset?

A dataset is basically a **collection of data**; it can be a list of strings, a list of integers or even a number of rows in a relational database.

- RDDs can contain any types of objects, including user-defined classes.
- An RDD is simply a **capsulation** around a **very large dataset**. In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.
- Under the hood, Spark will automatically **distribute the data** contained in RDDs across your cluster and **parallelize** the operations you perform on them.

We can do Transformations and Actions with the RDDs

Transformations

- Transformations are **operations** on RDDs which will return a **new RDD**.
- One of the most common transformations is filter which will return a new RDD with a subset of the data in the original RDD.

Actions

- Compute a result based on an RDD.
- One of the most popular Actions is first, which returns the first element in an RDD.

Spark RDD general workflow

- Generate initial RDDs from external data.
- Apply transformations.
- Launch actions.

Spark Context

- SparkContext is the entry point of Spark functionality. The most important step of any Spark driver application is to generate SparkContext.
- It allows your Spark Application to access Spark Cluster with the help of Resource Manager. The resource manager can be one of these three- Spark Standalone, YARN, Apache Mesos.
- A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.
- Only one SparkContext may be active per JVM. You must stop() the active SparkContext before creating a new one. This limitation may eventually be removed



Run Quick Start Docker

sudo docker run --hostname=quickstart.cloudera --privileged=true -t -v
/home/tatheen/Desktop/dataset:/user/cloudera/shared -i -p 8889:8888 -p
7180:7181 cloudera/quickstart /usr/bin/docker-quickstart

@quickstart:/	
File Edit View Search Terminal Help	
<mark>tatheer@tatheer:~\$</mark> sudo docker runhostname=quickstart.clouderaprivileged=t -v /home/tatheer/Desktop/dataset:/user/cloudera/shared -i -p 8889:8888 -p 7180: loudera/quickstart /usr/bin/docker-quickstart	rue -t 7181 c

Start Pyspark

pyspark

```
[root@quickstart /]# pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
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/Stati
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/Stati
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
20/05/25 09:40:32 INFO spark.SparkContext: Running Spark version 1.6.0
20/05/25 09:40:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
```



Create RDD

- Take an existing collection in your program and pass it to SparkContext's parallelize method.
- All the elements in the collection will then be copied to form a **distributed dataset** that can be operated on in **parallel**.
- Very handy to create an RDD with little effort.

```
inputIntegers = list(range(1,6))
```

integerRDD = sc.parallelize(inputIntegers)

```
>>> inputIntegers = list(range(1,6))
>>> integerRDD = sc.parallelize(inputIntegers)
>>> integerRDD
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
```

Load RDD

Create an input directory in hdfs user folder and upload the input file

lines =

sc.textFile("hdfs://quickstart.cloudera:8020/user/input/uppercase.text")

← → C ③ localhost:8889/filebrowser/view=/user#/user/input					0-	, ☆ 1	:
化 🖓 Query Editors 🗸 🛛 Data Browsers 🗸 Workflows 🗸	Search Security 🗸				💷 😋 🗸	⊗ ≋	۵
E File Browser							
Search for file name Actions V X Move to trash V					⊕ Upload ∨	O New	~
番 Home / user / input 🖋					 History 	🖻 Trash	
hame	🔶 Size	User	Group	Permissions	Date		\$
■ 2		hdfs	supergroup	drwxr-xr-x	May 25, 2020	06:43 AM	
		cloudera	supergroup	drwxr-xr-x	May 25, 2020	06:46 AM	
output		root	supergroup	drwxr-xr-x	vxr-xr-x May 25, 2020 06:46 Al		
uppercase.text	4.1 KB	cloudera	supergroup	-rw-rr	w-rr May 25, 2020 06:4		

Save File

lines.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/input/output")

\leftrightarrow \rightarrow C (i) localhost:8889/filebrowser/view=/user#/user/input/outp	ut					07	☆	T	:
🕂 🕹 🖓 Query Editors 🗸 🛛 Data Browsers 🗸 🛛 Workflows	∽ Search Security ∽			•		0° ~	0	 ≈	۲
File Browser									
Search for file name Actions V Move to trash	ſ				⊙ U	pload 🗸	O N	ew 🗸	
备 Home / user / input / output 🖋						listory	聞 Tr	ash	
🔶 Name	Size	User	Group	Permissions	Date				¢
♦ Name In 1	🔶 Size	♦ User cloudera	Group supergroup	Permissions drwxr-xr-x	Date May	25, 2020 (06:46 A	M	\$
Image: State	🔶 Size	User Cloudera root	Group supergroup supergroup	Permissions drwxr-xr-x drwxr-xr-x	Date May May	25, 2020 (25, 2020 (06:46 A	M	\$
Image: Name Image: J Image: Name Image: J Image: Success	Size 0 bytes	User cloudera root root	 Group supergroup supergroup supergroup 	Permissions drwxr-xr-x drwxr-xr-x -rw-r-r	Date May May May	25, 2020 (25, 2020 (25, 2020 (06:46 A 06:46 A 06:46 A	M M M	
Name J	Size 0 bytes 2.1 KB	 User cloudera root root root 	Group supergroup supergroup supergroup supergroup	 Permissions drwxr-xr-x drwxr-xr-x -rw-r-r -rw-rr 	Date May May May May	25, 2020 (25, 2020 (25, 2020 (25, 2020 (06:46 A 06:46 A 06:46 A 06:46 A	M M M M	\$
Image: Name Image: Jack State Image: Jack State <td>Size O bytes 2.1 KB 2.0 KB</td> <td>User Cloudera root root root root root root root</td> <td>Group supergroup supergroup supergroup supergroup supergroup supergroup</td> <td> Permissions drwxr-xr-x drwxr-xr-x -rw-r-r -rw-r-r -rw-r-r -rw-r-r </td> <td>Date May May May May</td> <td>25, 2020 (25, 2020 (25, 2020 (25, 2020 (25, 2020 (</td> <td>06:46 A) 06:46 A) 06:46 A) 06:46 A) 06:46 A)</td> <td>M M M M</td> <td>\$</td>	Size O bytes 2.1 KB 2.0 KB	User Cloudera root root root root root root root	Group supergroup supergroup supergroup supergroup supergroup supergroup	 Permissions drwxr-xr-x drwxr-xr-x -rw-r-r -rw-r-r -rw-r-r -rw-r-r 	Date May May May May	25, 2020 (25, 2020 (25, 2020 (25, 2020 (25, 2020 (06:46 A) 06:46 A) 06:46 A) 06:46 A) 06:46 A)	M M M M	\$

Spark Transformations

- Filter()
 - Takes in a function and returns an RDD formed by selecting those elements which pass the filter function.
 - Can be used to remove some invalid rows to clean up the input RDD or just get a subset of the input RDD based on the filter function.

- Map()
 - Takes in a function and passes each element in the input RDD through the function, with the result of the function being the new value of each element in the resulting RDD.
 - The return type of the map function is not necessary the same as its input type.

Create a spark program to read data from airport.text file and find all airports located in a country.

Upload airports.text in user/input

1	R H	ome / user / input 🖌					• History 💼 Trash
	+	Name	▲ Size	User	Group	Permissions	Date
	-	t		hdfs	supergroup	drwxr-xr-x	May 25, 2020 09:20 AM
	-			cloudera	supergroup	drwxr-xr-x	May 25, 2020 09:59 AM
		airports.text	88.7 KB	cloudera	supergroup	-rw-rr	May 25, 2020 09:59 AM
	-	output		root	supergroup	drwxr-xr-x	May 25, 2020 09:25 AM
		uppercase.text	4.1 KB	cloudera	supergroup	-rw-rr	May 25, 2020 09:24 AM

Create your python files i.e. AirportsInUsaSolution.py and Utils.py

AirportsInUsaSolution.py

```
import sys
from pyspark import SparkContext, SparkConf
import Utils
def splitComma(line: str):
    splits = Utils.COMMA_DELIMITER.split(line)
    return "{}, {}".format(splits[1], splits[2])
if __name__ == "__main__":
    conf = SparkCont().setAppName("airports").setMaster("local[*]")
    sc = SparkContext(conf = conf)
    airports = sc.textFile("hdfs://quickstart.cloudera:8020/user/input/
airports.text")
    airportsInUSA = airports.filter(lambda line :
    Utils.COMMA_DELIMITER.split(line)[3] == "\"United States\"")
    airportsNameAndCityNames = airportsInUSA.map(splitComma)
    airportsNameAndCityNames.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/input/user/input//user/input//user/input//user/input//user/input//user/input//user/input//user/input//user/input//user/input//user/input//user/input//airports_inusa.text")
```

Utils.py

import <mark>re</mark>

class Utils():

COMMA_DELIMITER = re.compile(''',(?=(?:[^"]*"[^"]*")*[^"]*\$)''')

Upload the two .py files in 'python' named directory

*	Home	/ user / python	1						▼ H	listory	🖻 Trash
	Name			Size	÷	User	÷	Group 🔶	Permissions	Date	÷
	t I					hdfs		supergroup	drwxr-xr-x	May 25,	2020 01:22 PM
his folde	e 🛌					cloudera		supergroup	drwxr-xr-x	May 25,	2020 01:22 PM
	AirportsIn	UsaProblem.py		62 bytes		cloudera		supergroup	-TW-TT	May 25,	2020 01:22 PM
	AirportsIn	UsaSolution.py		704 bytes		cloudera		supergroup	-TW-FF	May 25,	2020 01:22 PM
	ງ Utils.py			96 bytes		cloudera		supergroup	-TW-FF	May 25,	2020 01:22 PM

On pyspark shell

>>> exec('hdfs://quickstart.cloudera:8020/user/python/AirportsInUsaSolution.py'
)

Result:

Home / user / input / output2 / airports_in_usa.text /

	Name	🔶 Size	User	Group 🔶 Pern
	t		root	supergroup drwx
			root	supergroup drwx
0	_SUCCESS	0 bytes	root	supergroup -rw-r
0	part-00000	412 bytes	root	supergroup -rw-r
D	part-00001	292 bytes	root	supergroup -rw-r

- Union()
 - Union operation gives us back an RDD consisting of the data from both input RDDs
 - If there are any duplicates in the input RDDs, the resulting RDD of Spark's union operation will contain duplicates as well.

Write the code in pyspark shell

```
>>> student1_marks = [("physics",85),("maths",75),("chemistry",95)]
>>> student2_marks = [("physics",95),("maths",55),("chemistry",45)]
>>> s1 = sc.parallelize(student1_marks)
>>> s2 = sc.parallelize(student2_marks)
>>> union=s1.union(s2).collect()
20/05/27 12:33:40 INFO spark.SparkContext: Starting job: collect at <stdin>:1
20/05/27 12:33:40 INFO scheduler.DAGScheduler: Got job 1 (collect at <stdin>:1)
with 4 output partitions
```

Result:

```
>>> union
[('physics', 85), ('maths', 75), ('chemistry', 95), ('physics', 95), ('maths',
55), ('chemistry', 45)]
>>>
```

• Join()

• This transformation joins two RDDs based on a common key.

```
[('physics', 85), ('maths', 75), ('chemistry', 95), ('physics', 95), ('maths',
55), ('chemistry', 45)]
>>> Subject_wise_marks = s1.join(s2)
>>> Subject_wise_marks.collect()
```

Result:

[('maths', (75, 55)), ('physics', (85, 95)), ('chemistry', (95, 45))]

- Intersection()
 - Intersection operation returns the common elements which appear in both input RDDs.
 - Intersection operation removes all duplicates including the duplicates from single RDD before returning the results.
 - Intersection operation is quite expensive since it requires shuffling all the data across partitions to identify common elements.

```
>>> Cricket_team = ["sachin","abhay","michael","rahane","david","ross","raj","r
ahul","hussy","steven","sourav"]
>>> Toppers = ["rahul","abhay","laxman","bill","steve"]
>>>
>>> cricketRDD = sc.parallelize(Cricket_team)
>>> toppersRDD = sc.parallelize(Toppers)
>>> toppercricketers = cricketRDD.intersection(toppersRDD)
>>> toppercricketers.collect()
```

Result:



- Distinct()
 - This transformation is used to get rid of any **ambiguities**. As the name suggest it picks out the lines from the RDD that are **unique**.
 - The distinct transformation is **expensive** because it requires shuffling all the data across partitions to ensure that we receive only one copy of each element.

```
>>> best_story = ["movie1","movie3","movie7","movie5","movie8"]
>>> best_direction = ["movie11","movie1","movie5","movie10","movie7"]
>>> best_screenplay = ["movie10","movie4","movie6","movie7","movie3"]
>>> story_rdd = sc.parallelize(best_story)
>>> direction_rdd = sc.parallelize(best_direction)
>>> screen_rdd = sc.parallelize(best_screenplay)
>>> total_nomination_rdd = story_rdd.union(direction_rdd).union(screen_rdd)
>>> total_nomination_rdd.collect()
20/05/27 12:51:30 INFO spark.SparkContext: Starting job: collect at <stdin>:1
```

Result:

['movie1', 'movie3', 'movie7', 'movie5', 'movie8', 'movie11', 'movie1', 'movie5 ', 'movie10', 'movie7', 'movie10', 'movie4', 'movie6', 'movie7', 'movie3']

UDF (User Defined Functions)

• UDF's provide a simple way to **add separate functions** into Spark that can be used during various transformation stages. UDF's are generally used to perform **multiple** tasks on Spark RDD's.

Using movie review data

A Home

Upload data (filename: u.user) in user/input

/ user / input

- +	Name	🔶 Size	÷	User	÷	Group	Permissic
	t			hdfs		supergroup	drwxr-xr-x
	• · · · · · · · · · · · · · · · · · · ·			cloudera		supergroup	drwxr-xr-x
	output			root		supergroup	drwxr-xr-x
0	u.user	22.1 KB		cloudera		supergroup	-rw-rr
D	uppercase.text	4.1 KB		cloudera		supergroup	-rw-rr

userRDD= sc.textFile("hdfs://quickstart.cloudera:8020/user/input/u.user")

userRDD.count() //will display number of users

ø

Create two functions 'parse_N_calculate_age() and age_group() in pyspark shell to divide users into age group

```
>>> def parse_N_calculate_age(data):
... userid,age,gender,occupation,zip = data.split("|")
... return userid, age_group(int(age)),gender,occupation,zip,int(age)
```

••••	4.5	
>>>	der	age_group(age):
		if age < 10 :
		return '0-10'
		elif age < 20:
		return '10-20'
		elif age < 30:
		return '20-30'
		elif age < 40:
		return '30-40'
		elif age < 50:
		return '40-50'
		elif age < 60:
		return '50-60'
		elif age < 70:
		return '60-70'
		elif age < 80:
		return '70-80'
		else :
		return '80+'

. . .

To perform analysis on people in age group 20-30

```
data_with_age_bucket = userRDD.map(parse_N_calculate_age)
RDD_20_30 = data_with_age_bucket.filter(lambda line : '20-30' in
line)
```

Let's count the number users by their profession in the given age_group 20-30

```
freq = RDD_20_30.map(lambda line : line[3]).countByValue()
dict(freq)
```

Result:

```
>>> dict(freq)
{u'administrator': 19, u'lawyer': 4, u'healthcare': 4, u'marketing': 5, u'execu
tive': 7, u'doctor': 2, u'scientist': 8, u'student': 116, u'technician': 12, u'
librarian': 11, u'programmer': 30, u'salesman': 2, u'homemaker': 3, u'engineer'
: 23, u'none': 2, u'artist': 12, u'writer': 14, u'entertainment': 8, u'other':
38, u'educator': 12}
```

count the number of movie users in the same age group based on gender

```
age_wise = RDD_20_30.map (lambda line : line[2]).countByValue()
```

```
dict(age_wise)
```

Result:

```
>>> dict(age_wise)
{u'M': 247, u'F': 85}
>>>
```

Accumulators and Broadcast Variables

For **parallel processing**, Apache Spark uses **shared variables**. A copy of shared variable goes on each node of the cluster when the driver sends a task to the executor on the cluster, so that it can be used for performing tasks.

There are two types of shared variables supported by Apache Spark -

- 1.) Broadcast 2.) Accumulator
- Accumulators
 - Accumulators are variables that are used for aggregating information across the executors. For, example we can calculate how many records are corrupted or count events that occur during job execution for debugging purposes.
 - Using Accumulators for outlier detection in the above movie dataset. We are assuming that anyone who falls into age group 80+ is outlier and marked as over_age and anyone falling into age group 0-10 is also an outlier and marked as under_age.

```
Under_age = sc.accumulator(0)
Over_age = sc.accumulator(0)
```

Create a function outliers()

>>>	def	outliers(data):
		global Over_age, Under_age
		age_grp= data[1]
		if(age_grp == "70-80"):
		Over_age +=1
		if(age_grp == "0-10"):
• • •		Under_age +=1
• • •		return data

df = data_with_age_bucket.map(outliers).collect()

Check how many users are underage and overage

Under_age.value

Over_age.value

Result:



Broadcast Variables

- Broadcast variables are read-only shared variables that are cached and available on all nodes in a cluster in-order to access or use by the tasks.
- Spark broadcasts the common data (reusable) needed by tasks within each stage. The broadcasted data is cache in serialized format and deserialized before executing each task.

Example:

```
states = [("NY" ,"New York"),("CA" ,"California"),("FL" ,"Florida")]
countries =[("USA" ,"United States of America"),("IN" ,"India")]
bstates = sc.broadcast(states)
bcountries = sc.broadcast(countries)
data = [("James","Smith","USA","CA"), ("Michael" ,"Rose"
,"USA","NY"), ("Robert","Williams","USA","CA"), ("Maria" ,"Jones"
,"USA" ,"FL")]
rdd = sc.parallelize(data)
sc.parallelize(data[2]).collect()
```

Result:



Basic Count programming using flatMap

- flatMap is a transformation to create an RDD from an existing RDD.
- It takes each element from an existing RDD and it can produce 0, 1 or many outputs for each element.



Write code in pyspark shell

```
myfile =
```

sc.textFile("hdfs://quickstart.cloudera:8020/user/input/uppercase.te
xt")

counts = myfile.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda v1,v2: v1 + v2)

counts.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/input/ou
put_count")

Result:

\leftrightarrow \rightarrow C (i) localhost:8889/filebrowser/view=/user/input#/user/input/oup	ut_count				07	☆ 🚺	:
化 😂 🐐 Query Editors 🗸 🛛 Data Browsers 🗸 🛛 Workflows 🗸	Search Security ~				💷 🕫 🗸	8	•
File Browser							
Search for file name Actions V X Move to trash V Home / user / input / ouput_count V					⊙ Upload ∨ ▼ History	O New ∨ ⊞ Trash	•
🔷 🔶 Name	♦ Size	User	Group	Permissions	Date		÷.
■ 2		cloudera	supergroup	drwxr-xr-x	May 26, 2020 (09:00 AM	
		hdfs	supergroup	drwxr-xr-x	May 26, 2020 (09:01 AM	
	0 bytes	hdfs	supergroup	-rw-rr	May 26, 2020 (09:01 AM	
part-00000	2.8 KB	hdfs	supergroup	-rw-rr	May 26, 2020 (09:01 AM	
part-00001	2.9 KB	hdfs	supergroup	-rw-rr	May 26, 2020 (09:01 AM	

Home / user / input / ouput_count / part-00000

(u'', 7) (u'operations', 1) (u'all', 1) (u'proposed', 1) (u'Union', 1) (u'ver', 1) (u'rest', 1)