DEV 361 - Build & Monitor Apache Spark Applications

Version 5.0 – November 2015

Using This Guide

Icons Used in the Guide

This lab guide uses the following ions to draw attention to different types of information:

Note: Additional information that will clarify something, provide details, or help you avoid mistakes.

CAUTION: Details you **must** read to avoid potentially serious problems.

Q&A: A question posed to the learner during a lab exercise.

Try This! Exercises you can complete after class (or during class if you finish a lab early) to strengthen learning.

Lab Files

Here is a description of the materials (data files, licenses etc.) that are supplied to support the labs. These materials should be downloaded to your system before beginning the labs.

DEV361Data.zip	This file contains the dataset used for the Lab Activities. • sfpd.csv • J_AddCat.csv • J_AddDist.csv

DEV361LabFiles.zip	This .zip file contains all the files that you will need for the Labs:
	• Lab 4:Work with Pair RDD. It contains solutions files in Scala and Python.
	  ○ Lab4.txt (Scala)
	  ○ Lab4_py.txt (Python)
	• Lab 5: Work with DataFrames.
	  ○ Lab5.txt (Scala)

	○ Lab5_py.txt (Python) • Supplemental Lab ○ Lab5_4.zip (Zip file that will create folder structure required to build the application in SBT or in Maven for lab 5.4) ○ Lab5Supplemental.txt ○ resolution2015.scala ○ resolution2015.py ○ resolutions.sbt

Note: Refer to “Connect to MapR Sandbox or AWS Cluster” for instructions on connecting to the Lab environment.

Lesson 4 - Work with Pair RDD

Lab Overview

In this activity, you will load SFPD data from a CSV file. You will create pair RDD and apply pair RDD operations to explore the data.

Exercise	Duration
4.1: Load data into Apache Spark **Explore data in Apache Spark**	15 min
4.2 Create & Explore Pair RDD	25 min
4.3: Explore partitioning	15 min

Scenario

Our dataset is a .csv file that consists of SFPD incident data from SF OpenData (https://data.sfgov.org/). For each incident, we have the following information:

Field	Description	Example Value
IncidentNum	Incident number	150561637
Category	Category of incident	NON-CRIMINAL
Descript	Description of incident	FOUND_PROPERTY
DayOfWeek	Day of week that incident occurred	Sunday
Date	Date of incident	6/28/15
Time	Time of incident	23:50
PdDistrict	Police Department District	TARAVAL

Resolution	Resolution	NONE
Address	Address	1300_Block_of_LA_PLAYA_ST
X	X-coordinate of location	-122.5091348
Y	Y-coordinate of location	37.76119777
PdID	Department ID	15056163704013

The dataset has been modified to decrease the size of the files and also to make it easier to use. We use this same dataset for all the labs in this course.

Set up for the Lab

Download the files DEV361Data.zip and DEV361LabFiles.zip to your machine.

1. Copy DEV361Data.zip file to your home directory.

```
scp DEV361Data.zip <username>@node-ip:/user/<username>/data.zip
```

For example, if you are logged in as user01 and the IP address is 192.0.0.1

```
scp DEV361Data.zip user01@192.0.0.1:/user/user01/data.zip
```

2. Unzip data.zip

```
unzip data.zip
ls /user/<username>/data
```

You should see the data files here (sfpd.csv).

3. Copy DEV361LabFiles.zip to your home directory.

```
scp DEV361LabFiles.zip <username>@node-ip:/user/<username>
```

Lab 4.1: Load Data into Apache Spark

Objectives

- Launch the Spark interactive shell
- Load data into Spark
- Explore data in Apache Spark

4.1.1 Launch the Spark Interactive Shell

The Spark interactive shell is available in Scala or Python.

Note: All instructions here are for Scala.

1. To launch the Interactive Shell, at the command line, run the following command:

```
/opt/mapr/spark/spark-<version>/bin/spark-shell
```

Note: To find the Spark version

```
ls /opt/mapr/spark
```

Note: To quit the Scala Interactive shell, use the command

```
Exit
```

4.1.2. Load Data into Spark

The data we want to load is in the auctiondata.csv file. To load the data we are going to use the `SparkContext` method `textFile`. The SparkContext is available in the interactive shell as the variable **`sc`**. We also want to split the file by the separator ",".

1. We define the mapping for our input variables. While this isn't a necessary step, it makes it easier to refer to the different fields by names.

```
val IncidntNum = 0
val Category = 1
val Descript = 2
val DayOfWeek = 3
val Date = 4
val Time = 5
val PdDistrict = 6
val Resolution = 7
val Address = 8
val X = 9
val Y = 10
val PdId = 11
```

2. To load data into Spark, at the Scala command prompt:

```
val sfpdRDD = sc.textFile("/pathtofile/sfpd.csv").map(_.split(","))
```

Caution! If you do not have the correct path to the file sfpd.csv, you will get an error when you perform any actions on the RDD.

If you copied the files as per the directions, and you are logged in as user01,then the path to the file is:

```
/user/user01/data/sfpd.csv
```

Lab 4.1.3 Explore data using RDD operations

What transformations and actions would you use in each case? Complete the command with the appropriate transformations and actions.

1. How do you see the first element of the inputRDD (sfpdRDD)?

   `sfpdRDD.`_____

2. What do you use to see the first 5 elements of the RDD?

   `sfpdRDD.`_____

3. What is the total number of incidents?

   `val totincs = sfpdRDD.`_____

   _____

4. What is the total number of distinct resolutions?

   `val totres = sfpdRDD.`_____

   _____

5. List the PdDistricts.

   `val districts = sfpdRDD.`_____

   _____

Lab 4.2 Create & Explore PairRDD

In the previous activity we explored the data in the sfpdRDD. We used RDD operations. In this activity, we will create pairRDD to find answers to questions about the data.

Objectives

- Create pairRDD & apply pairRDD operations
- Join pairRDD

Lab 4.2.1. Create pair RDD & apply pair RDD operations

1. Which five districts have the highest incidents?

Note: This is similar to doing a word count. First, use the **`map`** transformation to create a pair RDD with the key being the field on which you want to count. In this case, the key would be PdDistrict and the value is "1" for each count of the district.

To get the total count, use **`reduceByKey((a,b)=>a+b)`**.

To find the **top** 5, you have to do a sort in descending. However, sorting on the result of the reduceByKey will sort on the District rather than the count. Apply the **`map`** transformation to create a pairRDD with the count being the key and the value being the district.

Then use **`sortByKey`**(false) to specify descending order.

To get the top 5, use **`take(5)`**.

Note that this is one way of doing it. There may be other ways of achieving the same result.

a. Use a **`map`** transformation to create pair RDD from sfpdRDD of the form: [(PdDistrict, 1)]

---

b. Use **`reduceByKey((a,b)=>a+b)`** to get the count of incidents in each district. Your result will be a pairRDD of the form [(PdDistrict, count)]

---

c. Use map again to get a pairRDD of the form [(count, PdDistrict)]

---

d. Use **`sortByKey(false)`** on [(count, PdDistrict)]

e. Use **`take(5)`** to get the top 5.

_____

2. Which five addresses have the highest incidents?
   a. Create pairRDD (map)
   b. Get the count for key (reduceByKey)
   c. Pair RDD with key and count switched (map
   d. Sort in descending order (sortByKey
   e. Use **`take(5)`** to get the top 5

_____

_____

3. What are the top **three** categories of incidents?

_____

_____

4. . What is the count of incidents by district?

_____

_____

Caution! For large datasets, don't use countByKey.

Note: Find the answers and the solutions to the questions at the end of Lab 4. The solutions in Scala and Python are provided. You can also refer to the files Lab4_2.txt (Scala); Lab4_2_py.txt (Python)

Lab 4.2.2: Join Pair RDDs

This activity illustrates how joins work in Spark (Scala). There are two small datasets provided for this activity - J_AddCat.csv and J_AddDist.csv.

J_AddCat.csv - Category; Address			J_AddDist.csv - PdDistrict; Address		
1.	EMBEZZLEMENT	100_Block_of_JEFFERSON_ST	1.	INGLESIDE	100_Block_of_ROME_ST
2.	EMBEZZLEMENT	SUTTER_ST/MASON_ST	2.	SOUTHERN	0_Block_of_SOUTHPARK_AV
3.	BRIBERY	0_Block_of_SOUTHPARK_AV	3.	MISSION	1900_Block_of_MISSION_ST
4.	EMBEZZLEMENT	1500_Block_of_15TH_ST	4.	RICHMOND	1400_Block_of_CLEMENT_ST
5.	EMBEZZLEMENT	200_Block_of_BUSH_ST	5.	SOUTHERN	100_Block_of_BLUXOME_ST
6.	BRIBERY	1900_Block_of_MISSION_ST	6.	SOUTHERN	300_Block_of_BERRY_ST
7.	EMBEZZLEMENT	800_Block_of_MARKET_ST	7.	BAYVIEW	1400_Block_of_VANDYKE_AV
8.	EMBEZZLEMENT	2000_Block_of_MARKET_ST	8.	SOUTHERN	1100_Block_of_MISSION_ST
9.	BRIBERY	1400_Block_of_CLEMENT_ST	9.	TARAVAL	0_Block_of_CHUMASERO_DR
10.	EMBEZZLEMENT	1600_Block_of_FILLMORE_ST			
11.	EMBEZZLEMENT	1100_Block_of_SELBY_ST			
12.	BAD CHECKS	100_Block_of_BLUXOME_ST			
13.	BRIBERY	1400_Block_of_VANDYKE_AV			

Based on the data above, answer the following questions:

1. Given these two datasets, you want to find the type of incident and district for each address. What is one way of doing this? (HINT: An operation on pairs or pairRDDs)

   What should the keys be for the two pairRDDs?

2. What is the size of the resulting dataset from a join? Why?

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a join is the same as an inner join and only keys that are present in both RDDs are output.

3. If you did a right outer join on the two datasets with Address/Category being the source RDD, what would be the size of the resulting dataset? Why?

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a right outer join results in a pair RDD that has entries for each key in the other pair RDD.

4. If you did a left outer join on the two datasets with Address/Category being the source RDD, what would be the size of the resulting dataset? Why?

Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a left outer join results in a pair RDD that has entries for each key in the source pair RDD.

5. Load each dataset into separate pairRDDs with “address” being the key.

Note: once you load the text file, split on the “,” and then apply the **`map`** transformation to create a pairRDD where address is the first element of the two-tuple.

```
val catAdd = sc._____
```

```
_____

val distAdd = sc._____

_____
```

6. List the incident category and district for those addresses that have both category and district information. Verify that the size estimated earlier is correct.

```
val catJdist = _____

catJdist.collect or catJdist.take(10)
```

For the size:

```
catJdist2._____
```

7. List the incident category and district for all addresses irrespective of whether each address has category and district information.

```
val catJdist1 = _____

catJdist1.collect or catJdist.take(10)
```

For the size:

```
catJdist2._____
```

8. List the incident district and category for all addresses irrespective of whether each address has category and district information. Verify that the size estimated earlier is correct.

```
val catJdist2 = _____

catJdist2.collect or catJdist.take(10)
```

For the size:

```
catJdist2._____
```

Lab 4.3 Explore Partitioning

In this activity we see how to determine the number of partitions, the type of partitioner and how to specify partitions in a transformation.

Objective

- Explore partitioning in RDDs

Lab 4.3.1: Explore partitioning in RDDs

Note To find partition size: rdd.partitions.size (Scala); rdd.getNumPartitions() (in Python)

To determine the partitioner: rdd.partitioner (Scala);

1. How many partitions are there in the sfpdRDD?

   sfpdRDD._____

2. How do you find the type of partitioner for sfpdRDD?

   sfpdRDD._____

3. Create a pair RDD.

   ```
   val incByDists =
   sfpdRDD.map(incident=>(incident(PdDistrict),1)).reduceByKey((x,y)
   =>x+y)
   ```

   How many partitions does incByDists have?

   incByDists._____

   What type of partitioner does incByDists have?

   incByDists._____

Q: Why does incByDists have that partitioner?

A: reduceByKey automatically uses the Hash Partitioner

4. Add a map

   ```
   val inc_map = incByDists.map(x=>(x._2,x._1))
   ```

   How many partitions does inc_map have? _____

   inc_map._____

   What type of partitioner does incByDists have? _____

   inc_map._____

Q: Why does inc_map not have the same partitioner as its parent?

A: Transformations such as map() cause the new RDD to forget the parent's partitioning information because a map() can modify the key of each record.

5. Add a sortByKey

   ```
   val inc_sort = inc_map.sortByKey(false)
   ```

   What type of partitioner does inc_sort have? _____

   inc_sort._____

Q: Why does inc_sort have this type of partitioner?

A: This is because sortByKey will automatically result in a range partitioned RDD..

6. Add groupByKey

```
val inc_group =
sfpdRDD.map(incident=>(incident(PdDistrict),1)).groupByKey()
```

What type of partitioner does inc_group have? _____

```
inc_group._____
```

Q: Why does inc_group have this type of partitioner?

A: This is because groupByKey will automatically result in a hash partitioned RDD..

7. Create two pairRDD

```
val catAdd =
sc.textFile("/user/user01/data/J_AddCat.csv").map(x=>x.split(",")
).map(x=>(x(1),x(0)))

val distAdd =
sc.textFile("/user/user01/data/J_AddDist.csv").map(x=>x.split(","
)).map(x=>(x(1),x(0)))
```

8. You can specify the number of partitions when you use the join operation.

```
val catJdist = catAdd.join(distAdd,8)
```

How many partitions does the joined RDD have? _____

```
catJdist._____
```

What type of partitioner does catJdist have? _____

```
catJdist._____
```

Note: A join will automatically result in a hash partitioned RDD.

Answers

Lab 4.1.3. Explore data using RDD operations

1. `Array[String] = Array(150599321, OTHER_OFFENSES, POSSESSION_OF_BURGLARY_TOOLS, Thursday, 7/9/15, 23:45, CENTRAL, ARREST/BOOKED, JACKSON_ST/POWELL_ST, -122.4099006, 37.79561712, 15059900000000)`
2. `Array[Array[String]] = Array(Array(150599321, OTHER_OFFENSES, POSSESSION_OF_BURGLARY_TOOLS, Thursday, 7/9/15, 23:45, CENTRAL, ARREST/BOOKED, JACKSON_ST/POWELL_ST, -122.4099006, 37.79561712, 15059900000000), Array(156168837, LARCENY/THEFT, PETTY_THEFT_OF_PROPERTY, Thursday, 7/9/15, 23:45, CENTRAL, NONE, 300_Block_of_POWELL_ST, -122.4083843, 37.78782711, 15616900000000), Array(150599321, OTHER_OFFENSES, DRIVERS_LICENSE/SUSPENDED_OR_REVOKED, Thursday, 7/9/15, 23:45, CENTRAL, ARREST/BOOKED, JACKSON_ST/POWELL_ST, -122.4099006, 37.79561712, 15059900000000), Array(150599224, OTHER_OFFENSES, DRIVERS_LICENSE/SUSPENDED_OR_REVOKED, Thursday, 7/9/15, 23:36, PARK, ARREST/BOOKED, MASONIC_AV/GOLDEN_GATE_AV, -122.4468469, 37.77766882, 15059900000000), Array(156169067, LARCENY/THEFT, GRAND_THEFT_F...`
3. 383775
4. 17
5. `Array[String] = Array(INGLESIDE, SOUTHERN, PARK, NORTHERN, MISSION, RICHMOND, TENDERLOIN, BAYVIEW, TARAVAL, CENTRAL)`

Lab 4.2.1. Create pair RDD & apply pair RDD operations

1. `Array((73308,SOUTHERN), (50164,MISSION), (46877,NORTHERN), (41914,CENTRAL), (36111,BAYVIEW))`
2. `Array[(Int, String)] = Array((10852,800_Block_of_BRYANT_ST), (3671,800_Block_of_MARKET_ST), (2027,1000_Block_of_POTRERO_AV), (1585,2000_Block_of_MISSION_ST), (1512,16TH_ST/MISSION_ST))`
3. `Array((96955,LARCENY/THEFT), (50611,OTHER_OFFENSES), (50269,NON-CRIMINAL))`
4. `Map(SOUTHERN -> 73308, INGLESIDE -> 33159, TENDERLOIN -> 30174, MISSION -> 50164, TARAVAL -> 27470, RICHMOND -> 21221, NORTHERN -> 46877, PARK -> 23377, CENTRAL -> 41914, BAYVIEW -> 36111)`

Lab 4.2.2. Join Pair RDDs

```
6. Array[(String, (String, String))] =
   Array((1400_Block_of_CLEMENT_ST,(BRIBERY,RICHMOND)),
   (100_Block_of_BLUXOME_ST,(BAD CHECKS,SOUTHERN)),
   (1400_Block_of_VANDYKE_AV,(BRIBERY,BAYVIEW)),
   (0_Block_of_SOUTHPARK_AV,(BRIBERY,SOUTHERN)),
   (1900_Block_of_MISSION_ST,(BRIBERY,MISSION)))

      count =5

7. Array[(String, (String, Option[String]))] =
   Array((1600_Block_of_FILLMORE_ST,(EMBEZZLEMENT,None)),
   (1400_Block_of_CLEMENT_ST,(BRIBERY,Some(RICHMOND))),
   (100_Block_of_BLUXOME_ST,(BAD CHECKS,Some(SOUTHERN))),
   (1100_Block_of_SELBY_ST,(EMBEZZLEMENT,None)),
   (1400_Block_of_VANDYKE_AV,(BRIBERY,Some(BAYVIEW))),
   (100_Block_of_JEFFERSON_ST,(EMBEZZLEMENT,None)),
   (SUTTER_ST/MASON_ST,(EMBEZZLEMENT,None)),
   (0_Block_of_SOUTHPARK_AV,(BRIBERY,Some(SOUTHERN))),
   (800_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
   (2000_Block_of_MARKET_ST,(EMBEZZLEMENT,None)),
   (1500_Block_of_15TH_ST,(EMBEZZLEMENT,None)),
   (200_Block_of_BUSH_ST,(EMBEZZLEMENT,None)),
   (1900_Block_of_MISSION_ST,(BRIBERY,Some(MISSION))))

count = 13

8. Array[(String, (Option[String], String))] =
   Array((300_Block_of_BERRY_ST,(None,SOUTHERN)),
   (1400_Block_of_CLEMENT_ST,(Some(BRIBERY),RICHMOND)),
   (100_Block_of_BLUXOME_ST,(Some(BAD CHECKS),SOUTHERN)),
   (1400_Block_of_VANDYKE_AV,(Some(BRIBERY),BAYVIEW)),
   (0_Block_of_SOUTHPARK_AV,(Some(BRIBERY),SOUTHERN)),
   (0_Block_of_CHUMASERO_DR,(None,TARAVAL)),
   (1100_Block_of_MISSION_ST,(None,SOUTHERN)),
   (100_Block_of_ROME_ST,(None,INGLESIDE)),
   (1900_Block_of_MISSION_ST,(Some(BRIBERY),MISSION)))

      count = 9
```

Lab 4.3.1: Explore Partitioning in Pair RDDs

1. The answer to this will vary depending on your environment. In our training environment, sfpdRDD has 2 partitions.

2. No partitioner

3. In the training environment - 2 partitions; Hash Partitioner

4. Same number of partitions; type of partitioner = none
5. Range partitioner
6. Hash partitioner
8. 8; Hash Partitioner

Solutions

Lab 4.1.3 – Scala

Note: Solutions are also in the file **Lab4.txt** from which you can copy and paste into the Interactive shell.

```
1. sfpdRDD.first()
2. sfpdRDD.take(5)
3. val totincs = sfpdRDD.count()
4. val totres = sfpdRDD.map(inc=>inc(Resolution)).distinct.count
5. val dists = sfpdRDD.map(inc=>inc(PdDistrict)).distinct
dists.collect
```

Lab 4.1.3 – Python

To launch the Python shell,

```
$ opt/mapr/spark/spark-<version>/bin/pyspark
```

Note: Solutions are also in the file **Lab4_py.txt** from which you can copy and paste into the Interactive shell.

To map input variables:

```
IncidntNum = 0
Category = 1
Descript = 2
DayOfWeek = 3
Date = 4
Time = 5
PdDistrict = 6
Resolution = 7
Address = 8
X = 9
Y = 10
PdId = 11
```

To load the file:

```
sfpdRDD=sc.textFile("/path/to/file/sfpd.csv").map(lambda
line:line.split(","))
```

1. `sfpdRDD.first`
2. `sfpdRDD.take(5)`
3.
```
totincs = sfpdRDD.count()
print totincs
```
4.
```
totres = sfpdRDD.map(lambda
inc:inc[Resolution]).distinct().count()
print totres
```
5.
```
dists = sfpdRDD.map(lambda
inc:inc[PdDistrict]).distinct().collect()
print dists
```

Lab 4.2.1 - Scala

Note: Solutions are also in the file **Lab4.txt** from which you can copy and paste into the Interactive shell.

1.
```
val top5Dists =
sfpdRDD.map(incident=>(incident(PdDistrict),1)).reduceByKey((x,y)
=>x+y).map(x=>(x._2,x._1)) .sortByKey(false).take(3)
```
2.
```
val top5Adds =
sfpdRDD.map(incident=>(incident(Address),1)).reduceByKey((x,y)=>x
+y).map(x=>(x._2,x._1)) .sortByKey(false).take(5)
```
3.
```
val top3Cat =
sfpdRDD.map(incident=>(incident(Category),1)).reduceByKey((x,y)=>
x+y).map(x=>(x._2,x._1)) .sortByKey(false).take(3)
```
4.
```
val num_inc_dist =
sfpdRDD.map(incident=>(incident(PdDistrict),1)).countByKey()
```

Lab 4.3.1 - Python

Note: Solutions are also in the file **Lab4_py.txt** from which you can copy and paste into the Interactive shell.

1.
```
top5Dists=sfpdRDD.map(lambda
incident:(incident[PdDistrict],1)).reduceByKey(lambda
x,y:x+y).map(lambda x:(x[2],x[1])).sortByKey(false).take(5)
```

```
   print top5Dist
```

```
2. top5Adds=sfpdRDD.map(lambda
   incident:(incident[Address],1)).reduceByKey(lambda
   x,y:x+y).map(lambda x:(x[2],x[1])).sortByKey(false).take(5)

   print top5Adds
```

```
3. top3Cat=sfpdRDD.map(lambda
   incident:(incident[Category],1)).reduceByKey(lambda
   x,y:x+y).map(lambda x:(x[2],x[1])).sortByKey(false).take(3)

   print top3Cat
```

```
4. num_inc_dist=sfpdRDD.map(lambda
   incident:(incident[PdDistrict],1)).countByKey()

   print num_inc_dist
```

Lab 4.2.2

1. You can use joins on pairs or pairRDD to get the information. The key for the pairRDD is the address.

2. A join is the same as an inner join and only keys that are present in both RDDs are output. If you compare the addresses in both the datasets, you find that there are five addresses in common and they are unique. Thus the resulting dataset will contain five elements. If there are multiple values for the same key, the resulting RDD will have an entry for every possible pair of values with that key from the two RDDs.

3. A right outer join results in a pair RDD that has entries for each key in the other pairRDD. If the source RDD contains data from J_AddCat.csv and the “other” RDD is represented by J_AddDist.csv, then since “other” RDD has 9 distinct addresses, the size of the result of a right outer join is 9.

4. A left outer join results in a pair RDD that has entries for each key in the source pairRDD. If the source RDD contains data from J_AddCat.csv and the “other” RDD is represented by J_AddDist.csv, then since “source” RDD has 13 distinct addresses, the size of the result of a left outer join is 13.

Lab 4.2.2 - Scala

Note: Solutions are also in the file **Lab4.txt** from which you can copy and paste into the Interactive shell.

```
5. val catAdd =
   sc.textFile("/user/user01/data/J_AddCat.csv").map(x=>x.split(",")).m
   ap(x=>(x(1),x(0)))
```

```
   val distAdd =
   sc.textFile("/user/user01/data/J_AddDist.csv").map(x=>x.split(",")).
   map(x=>(x(1),x(0)))
```

6. `val catJdist=catAdd.join(distAdd)`

   `catJDist.collect`

   `catJdist.count`

   `catJdist.take(10)`

7. `val catJdist1 = catAdd.leftOuterJoin(distAdd)`

   `catJdist1.collect`

   `catJdist.count`

8. `val catJdist2 = catAdd.rightOuterJoin(distAdd)`

   `catJdist2.collect`

   `catJdist2.count`

Lab 4.2.2 - Python

Note: Solutions are also in the file **Lab4_py.txt** from which you can copy and paste into the Interactive shell.

5. `catAdd=sc.textFile("/path/to/file/J_AddCat.csv").map(lambda x:x.split(",")).map(lambda x:(x[1],x[0]))`

   `distAdd=sc.textFile("/path/to/file/J_AddDist.csv").map(lambda x:x.split(",")).map(lambda x:(x[1],x[0]))`

6. `catJdist=catAdd.join(distAdd)`

   `catJDist.collect`

   `catJdist.count`

   `catJdist.take(10)`

7. `catJdist1 = catAdd.leftOuterJoin(distAdd)`

   `catJdist1.collect`

   `catJdist.count`

8. `catJdist2 = catAdd.rightOuterJoin(distAdd)`

   `catJdist2.collect`

   `catJdist2.count`

Lab 4.3.1. - Scala

```
1. sfpdRDD.partitions.size
2. sfpdRDD.partitioner
3. incByDists.partitions.size; incByDists.partitioner
4. inc_map.partitions.size; inc_map.partitioner
5. inc_sort.partitioner
6. inc_group.partitioner
7. incByDists.partitions.size
9. catJdist.partitions.size
   catjDist.partitioner
```

Lab 4.3.2 - Python

Note: Solutions are also in the file **Lab4_py.txt** from which you can copy and paste into the Interactive shell.

Lesson 5 - Work with DataFrames

Lab Overview

In this activity, you will load SFPD data from a CSV file. You will create pair RDD and apply pair RDD operations to explore the data.

Exercise	Duration
5.1: Create DataFrames using reflection	20 min
5.2: Explore data in DataFrames	20 min
5.3: Create and use User Defined Functions	20 min
5.4: Build a Standalone Application - OPTIONAL	30 min

Scenario

Our dataset is a .csv file that consists of SFPD incident data from SF OpenData (https://data.sfgov.org/). For each incident, we have the following information:

Field	Description	Example Value
IncidentNum (String)	Incident number	150561637
Category (String)	Category of incident	NON-CRIMINAL
Descript (String)	Description of incident	FOUND_PROPERTY
DayOfWeek (String)	Day of week that incident occurred	Sunday
Date (String)	Date of incident	6/28/15
Time (String)	Time of incident	23:50
PdDistrict (String)	Police Department District	TARAVAL

Resolution (String)	Resolution	NONE
Address (String)	Address	1300_Block_of_LA_PLAYA_ST
X (Float)	X-coordinate of location	-122.5091348
Y (Float)	Y-coordinate of location	37.76119777
PdID (String)	Department ID	15056163704013

We use this same dataset for all the labs in this course.

Set up for the Lab

Download the files DEV361Data.zip and DEV361LabFiles.zip to your machine.

1. Copy DEV360Data.zip file to your home directory.

```
scp DEV361Data.zip <username>@node-ip:/user/<username>/data.zip
```

2. Unzip data.zip

```
unzip data.zip
ls /user/<username>/data
```

You should see the data files here (sfpd.csv).

3. Copy DEV361LabFiles.zip to your home directory.

```
scp DEV361LabFiles.zip <username>@node-ip:/user/<username>
```

Lab 5.1: Create DataFrame Using Reflection

Objectives

- Launch the Spark interactive shell
- Create RDD
- Create DataFrame using reflection to infer schema

5.1.1 Launch the Spark Interactive Shell

The Spark interactive shell is available in Scala or Python.

Note: All instructions here are for Scala.

1. To launch the Interactive Shell, at the command line, run the following command:

```
/opt/mapr/spark/spark-<version>/bin/spark-shell
```

Note: To find the Spark version

```
ls /opt/mapr/spark
```

Note: To quit the Scala Interactive shell, use the command

```
exit
```

5.1.2. Create RDD

The data we want to load is in the file sfpd.csv. To load the data we are going to use the `SparkContext` method `textFile`. SparkContext is available in the interactive shell as the variable **`sc`**. We also want to split the file by the separator ",".

1. To load data into Spark, at the Scala command prompt:

```
val sfpdRDD = sc.textFile("/path to
file/sfpd.csv").map(_.split(","))
```

Caution! If you do not have the correct path to the file sfpd.csv, you will get an error when you perform any actions on the RDD.

Lab 5.1.3. Create DataFrame Using Reflection to Infer Schema

In this activity, you will create a DataFrame from the RDD created in the previous activity using Reflection to infer schema.

1. Import required classes.

```
import sqlContext._

import sqlContext.implicits._
```

2. Define Case class. The case class defines the table schema. You specify the name of the class, each field and type. Below is the list of fields and type.

IncidentNum (String)	Category (String)	Descript (String)	DayOfWeek (String)
Date (String)	Time (String)	PdDistrict (String)	Resolution (String)

Address (String) X (Float) Y (Float) PdId (String)

To define the case class Incidents: (Complete the statement below)

```
case class Incidents(incidentnum:String, category:String,
description:__________, dayofweek:__________, date:__________,
time:__________, pddistrict:__________, resolution:__________,
address:__________, X:__________, Y:__________, pdid:__________)
```

3. Convert RDD (sfpdRDD) into RDD of case objects (sfpdCase) using the map transformation to map the case class to every element in the RDD.

```
val sfpdCase = sfpdRDD.map(inc=>Incidents(inc(0), inc(1), inc(2),
inc(3), inc(4),inc(5), inc(6), inc(7), inc(8), inc(9).toFloat,
inc(10).toFloat, inc(11)))
```

4. Implicitly convert resulting RDD of case objects into a DataFrame. (**HINT:** Use to.DF() method on RDD)

```
val sfpdDF = ____________________________________
```

5. Register the DataFrame as a table called **sfpd**.

```
sfpdDF.____________________________________
```

Q: Why do we register the DataFrame as a table?

A: Registering a DataFrame as a table enables you to query it using SQL.

Lab 5.2 Explore Data in DataFrames

In this activity, you will use DataFrame operations and SQL to explore the data in the DataFrames. Use DataFrame operations or SQL queries to answer the questions below.

Note: Refer to the following links for more help on DataFrames.

https://spark.apache.org/docs/1.3.1/sql-programming-guide.html

https://spark.apache.org/docs/1.3.1/api/scala/index.html#org.apache.spark.sql.DataFrame

1. What are the top 5 districts with the most number of incidents? (**HINT:** Use `groupBy; count; sort`. You can use `show(5)` to show five elements.)

```
val incByDist = sfpdDF.____________________________________

____________________________________
```

HINT: You pass in the SQL query within the parenthesis. For example:
`sqlContext.sql("SELECT <column>, count(incidentnum) AS inccount FROM <DataFrame Table> GROUP BY <column> ORDER BY inccount <SORT ORDER> LIMIT <number of records>")`

```
val incByDistSQL = sqlContext.sql("_____
_____")
```

2. What are the top 10 resolutions?

```
val top10Res = sfpdDF._____
_____
val top10ResSQL = sqlContext.sql("_____
_____")
```

3. What are the top three categories of incidents?

```
val top3Cat = sfpdDF._____
_____
val top3CatSQL = sqlContext.sql("_____
_____")
```

4. Save the top 10 resolutions to a JSON file in the folder **/user/<username>/output**.

HINT: Use **`DF.toJSON.saveAsTextFile`**

NOTE: This method has been deprecated in **Spark 1.4**. Use
`DF.write.format("json").mode("<mode type>").save("<path to file>")`

https://spark.apache.org/docs/1.4.1/sql-programming-guide.html#generic-loadsave-functions

Caution! If the folder already exists, you will get an error. You need to delete the output directory first or add logic to remove the directory if it exists before saving.

```
top10ResSQL._____
```

To verify that the data was saved to the file:

```
cd /user/<username>/output
cat part-00000
```

Lab 5.3 Create and Use User Defined Functions

The date field in this dataset is a String of the form “mm/dd/yy”. We are going to create a function to extract the year from the date field. There are two ways to use user defined functions with Spark DataFrames. You can define it inline and use it within DataFrame operations or use it in SQL queries.

We can then find answers to questions such as:

What is the number of incidents by year?

Objectives

- Create and use UDF with DataFrames Operations
- Create and use UDF in SQL queries

Lab 5.3.1 Create and use UDF with DataFrame Operations

1. Create a function to extract the year from the date string.

NOTE: One way to extract the characters after the last '/' is:
`s.substring(s.lastIndexOf('/')+1)` where s is the string from which you wish to extract the characters.

The function shown below, extracts the characters after the last "/" to give the year

```
val getyy =  udf( _____
_____)
```

NOTE: Define a function "getyy" as a udf that accepts one argument - a string **s**. Within the function, extract the characters after the last '/' from the string s and returns the value.

2. What is the number of incidents by year?

```
val incbyyear = sfpdDF.groupBy(getyy(sfpdDF("date"))).count.show
```

Lab 5.3.2 Create and use UDF in SQL queries

1. Define a function that extracts characters after the last '/' from the string.

```
def getyear(s:String):String = {
val year = _____
year
}
```

Q: What do we need to do in order to use this function in SQL queries?

A: Register this function as a udf -use `sqlContext.udf.register`.

2. Register the function

```
sqlContext.udf.register.("functionname",functionname _)
```

NOTE: The second parameter to the register function can be used to define the udf inline.

If you have already defined the function, then this parameter consists of the function name followed by _. There is a space before the underscore. This is specific to Scala. The underscore (must have a space in between function and underscore) turns the function into a *partially applied function* that can be passed to register. The underscore tells Scala that we don't know the parameter yet but want the function as a value. The required parameter will be supplied later

3. Using the registered the udf in a SQL query, find the count of incidents by year.

```
val incyearSQL = sqlContext.sql("SELECT getyear(date),
count(incidentnum) AS countbyyear FROM sfpd GROUP BY
getyear(date) ORDER BY countbyyear DESC")
```

4. Find the category, address and resolution of incidents reported in 2014.

```
val inc2014 =
sqlContext.sql(._____________________________________________
```

5. Find the addresses and resolution of VANDALISM incidents in 2015.

```
  val van2014 = sqlContext.sql("_____________________________
_____________________________________________________________")
```

Try creating other functions. For example, a function that extracts the month, and use this to see which month in 2015 has the most incidents.

Lab 5.4 Build a Standalone Application (Optional)

Now that you have explored DataFrames and created simple user defined functions, build a standalone application using DataFrames that:

- Loads sfpd.csv
- Creates a DataFrame (inferred schema by reflection)
- Registers the DataFrame as a table
- Prints the top three categories to the console
- Finds the address, resolution, date and time for burglaries in 2015
- Saves this to a JSON file in a folder /<user home>/appoutput

High Level steps:

- Use Lab 5.1 to load the data
- Use Lab 5.2.1 to create the DataFrame and register as a table
- Refer to Lab 5.2.2 to get the top three categories
- Refer to Lab 5.3 to create a UDF that extracts the year
- Refer to Lab 5.2.2 to save to a JSON file

Answers

Lab 5.2

1.

```
pddistrict count
SOUTHERN   73308
MISSION    50164
NORTHERN   46877
CENTRAL    41914
BAYVIEW    36111
```

2.

```
[NONE,243538]
[ARREST/BOOKED,86766]
[ARREST/CITED,22925]
[PSYCHOPATHIC_CASE,8344]
[LOCATED,6878]
[UNFOUNDED,4551]
[COMPLAINANT_REFUSES_TO_PROSECUTE,4215]
……..
```

3.

```
[LARCENY/THEFT,96955]
[OTHER_OFFENSES,50611]
```

Lab 5.3.1

2.

```
scalaUDF(date) count
13             152830
14             150185
15             80760
```

Lab 5.3.2

3.

```
[13,152830]
[14,150185]
[15,80760]
```

4.

```
[ASSAULT,0_Block_of_UNITEDNATIONS_PZ,NONE,12/31/14]
[NON-CRIMINAL,400_Block_of_POWELL_ST,NONE,12/31/14]
[ASSAULT,700_Block_of_MONTGOMERY_ST,NONE,12/31/14]
[VANDALISM,FOLSOM_ST/SPEAR_ST,NONE,12/31/14]
[NON-CRIMINAL,2800_Block_of_LEAVENWORTH_ST,NONE,12/31/14]
[NON-CRIMINAL,3300_Block_of_WASHINGTON_ST,NONE,12/31/14]
[OTHER_OFFENSES,3300_Block_of_WASHINGTON_ST,NONE,12/31/14]
.....
```

5.

```
[VANDALISM,FOLSOM_ST/SPEAR_ST,NONE,12/31/14]
[VANDALISM,300_Block_of_LOUISBURG_ST,NONE,12/31/14]
[VANDALISM,900_Block_of_WOOLSEY_ST,NONE,12/31/14]
[VANDALISM,800_Block_of_KEARNY_ST,NONE,12/31/14]
[VANDALISM,900_Block_of_ELLSWORTH_ST,NONE,12/31/14]
[VANDALISM,2000_Block_of_BALBOA_ST,NONE,12/31/14]
[VANDALISM,500_Block_of_DUBOCE_AV,NONE,12/31/14]
[VANDALISM,0_Block_of_WALLER_ST,NONE,12/31/14]
.....
van2015 count = 3898
```

Solutions

Note: Solutions are also in the file **Lab5.txt** from which you can copy and paste into the Interactive shell.

Lab 5.1.2 – Scala

```
1. val sfpdRDD = sc.textFile("/path to
   file/sfpd.csv").map(_.split(","))
```

Lab 5.1.3 - Scala

```
1. import sqlContext._

   import sqlContext.implicits._

2. case class Incidents(incidentnum:String, category:String,
   description:String, dayofweek:String, date:String, time:String,
   pddistrict:String, resolution:String, address:String, X:Float,
   Y:Float, pdid:String)

3. val sfpdCase=sfpdRDD.map(inc=>Incidents(inc(0),inc(1),
   inc(2),inc(3),inc(4),inc(5),inc(6),inc(7),inc(8),inc(9).toFloat,inc(
   10).toFloat, inc(11)))

4. val sfpdDF=sfpdCase.toDF()

5. sfpdDF.registerTempTable("sfpd")
```

Lab 5.2 - Scala

```
1. val incByDist =
   sfpdDF.groupBy("pddistrict").count.sort($"count".desc).show(5)

   val topByDistSQL = sqlContext.sql("SELECT pddistrict,
   count(incidentnum) AS inccount FROM sfpd GROUP BY pddistrict ORDER
   BY inccount DESC LIMIT 5")

2. val top10Res =
   sfpdDF.groupBy("resolution").count.sort($"count".desc)

   top10Res.show(10)


   val top10ResSQL = sqlContext.sql("SELECT resolution,
   count(incidentnum) AS inccount FROM sfpd GROUP BY resolution ORDER
   BY inccount DESC LIMIT 10")
```

3.
```
val top3Cat =
sfpdDF.groupBy("category").count.sort($"count".desc).show(3)


val top3CatSQL=sqlContext.sql("SELECT category, count(incidentnum)
AS inccount FROM sfpd GROUP BY category ORDER BY inccount DESC LIMIT
3")
```

4.
```
top10ResSQL.toJSON.saveAsTextFile("/<userhome>/output")
```

Lab 5.3.1 - Scala

1.
```
val getyy=  udf((s:String)=>{val yy =
s.substring(s.lastIndexOf('/')+1)
yy})
```

2.
```
val incbyyear = sfpdDF.groupBy(getyy(sfpdDF("date"))).count.show
```

Lab 5.3.2 - Scala

1.
```
def getyear(s:String):String = {
    val year = s.substring(s.lastIndexOf('/')+1)
    year
    }
```

2.
```
sqlContext.udf.register("getyear",getyear _)
```

3.
```
val incyearSQL=sqlContext.sql("SELECT getyear(date),
count(incidentnum) AS countbyyear FROM sfpd GROUP BY getyear(date)
ORDER BY countbyyear DESC")
incyearSQL.collect.foreach(println)
```

4.
```
val inc2014 = sqlContext.sql("SELECT category,address,resolution,
date FROM sfpd WHERE getyear(date)='14'")
inc2014.collect.foreach(println)
```

5.
```
val van2015 = sqlContext.sql("SELECT category,address,resolution,
date FROM sfpd WHERE getyear(date)='15' AND category='VANDALISM'")
van2015.collect.foreach(println)
van2015.count
```

Lesson 6 - Monitor Spark Applications

Lab Overview

In this activity, you will load SFPD data from a CSV file. You will create RDDs, apply transformations and actions, view the lineage of an RDD and relate it to the Spark execution steps in the Spark Web UI.

Exercise	Duration
6.1: Using the Spark UI	15 min
6.2: Finding Spark System Properties	10 min

Lab 6.1: Using the Spark UI

In this activity, you will take a look at Spark execution components in the Spark UI.

Objectives

- Launch the Spark Interactive Shell
- Load data into Spark
- Use Spark Web UI

6.1.1 Launch the Spark Interactive Shell

The Spark interactive shell is available in Scala or Python.

Note: All instructions here are for Scala.

1. To launch the Interactive Shell, at the command line, run the following command:

```
/opt/mapr/spark/spark-<version>/bin/spark-shell
```

Note: To find the Spark version

```
ls /opt/mapr/spark
```

Note: To quit the Scala Interactive shell, use the command

```
Exit
```

6.1.2. Load Data into Spark

The data we want to load is in the auctiondata.csv file. To load the data we are going to use the `SparkContext` method `textFile`. The SparkContext is available in the interactive shell as the variable **`sc`**. We also want to split the file by the separator ",".

1. We define the mapping for our input variables. While this isn't a necessary step, it makes it easier to refer to the different fields by names.

```
val IncidntNum = 0
val Category = 1
val Descript = 2
val DayOfWeek = 3
val Date = 4
val Time = 5
val PdDistrict = 6
val Resolution = 7
val Address = 8
val X = 9
val Y = 10
val PdId = 11
```

2. To load data into Spark, at the Scala command prompt:

```
val sfpdRDD = sc.textFile("/pathtofile/sfpd.csv").map(_.split(","))
```

6.1.3. Use the Spark Web UI

We are going to apply transformations to the input RDD (sfpdRDD) and create intermediate RDD.

```
1. val resolutions = sfpdRDD.map(inc=>inc(Resolution)).distinct
```

2. At this point we have created an intermediate RDD that has all the resolutions. View the lineage of resolutions.

```
resolutions.toDebugString
```

   a. Identify the lines in the lineage that correspond to the input RDD (i.e. sfpdRDD).

_____

_____

   b. Can you identify which transformation creates a shuffle?

_____

c. If you performed an action on **resolutions**, how many stages would you expect this job to have? Why?

_____

_____

3. Now add an action.

```
val totres = resolutions.count
```

4. Launch the Spark Web UI. From a browser, go to http:// **<ip address>:4040**. In the Spark Web UI, find your job. Check the number of stages. Is this the same as your estimation based on the lineage graph? _____

   How long did the job take? _____

   Click on the job to go to the job details page. How long did each stage take?

   _____

   How many tasks are there in each stage?

   _____

5. Now cache resolutions, and run the count again so the RDD is cached. Then run collect.

```
resolutions.cache()
```

resolutions.count (to actually cache the RDD)

```
resolutions.count
```

6. Go to the Spark Web UI.

   a. Compare the time taken to run this count against the very first count. Is there a difference?

_____

b. How many stages does this job have? Were any stages skipped?

_____

c. How many tasks in each stage? Are there any skipped tasks?

_____

Lab 6.2: Find Spark System Properties

This activity describes where to find Spark system properties and ways to set these properties.

Objectives

- Find Spark system properties in the Spark Web UI

Lab 6.2.1. Find Spark system properties in the Spark Web UI

Use the Spark Web UI to answer the following:

1. Where can you find Spark system properties in the Spark Web UI?

_____

2. What value has been set for spark.executor.memory? _____

3. Is the Spark event log enabled? Which property determines this?_____

4. To what value is Spark master set? _____

5. What is the Spark scheduler mode? _____

6. Which property gives us the location of Java on the system?_____

Spark Properties: Spark properties control most of the application settings and are configured separately for each application. SparkConf allows you to configure some of the common properties through the set() method. You can set these properties directly on the SparkConf that is passed into the SparkContext.For example:

```
val conf = new SparkConf().setMaster("local[2]").setAppName("SFPD
Incidents").set("spark.executor.memory","1g")
val sc = new SparkContext(conf)
```

You can also load these properties when calling `spark-submit`.

Some examples of Spark properties are listed below.

- spark.executor.memory - This indicates the amount of memory to be used per executor. •

- spark.serializer - Class used to serialize objects that will be sent over the network. Since the default java serialization is quite slow, it is recommended to use the org.apache.spark.serializer.JavaSerializer class to get a better performance.
- spark.kryo.registrator - Class used to register the custom classes if we use the Kyro serialization
- spark.local.dir - locations that Spark uses as scratch space to store the map output files.

Environment Variables: Some of the Spark settings can be configured using the environment variables that are defined in the **`conf/spark-env.sh`** script file. These are machine specific settings, such as library search path, java path, etc.

Answers

Lab 6.1.3

```
2. resolutions.toDebugString
```

```
(2) MapPartitionsRDD[11] at distinct at <console>:25 []
 |  ShuffledRDD[10] at distinct at <console>:25 []
 +-(2) MapPartitionsRDD[9] at distinct at <console>:25 []
    |  MapPartitionsRDD[8] at map at <console>:25 []
    |  MapPartitionsRDD[2] at map at <console>:21 []
    |  /user/user01/data/sfpd.csv MapPartitionsRDD[1] at textFile at <console>:21 []
    |  /user/user01/data/sfpd.csv HadoopRDD[0] at textFile at <console>:21 []
```

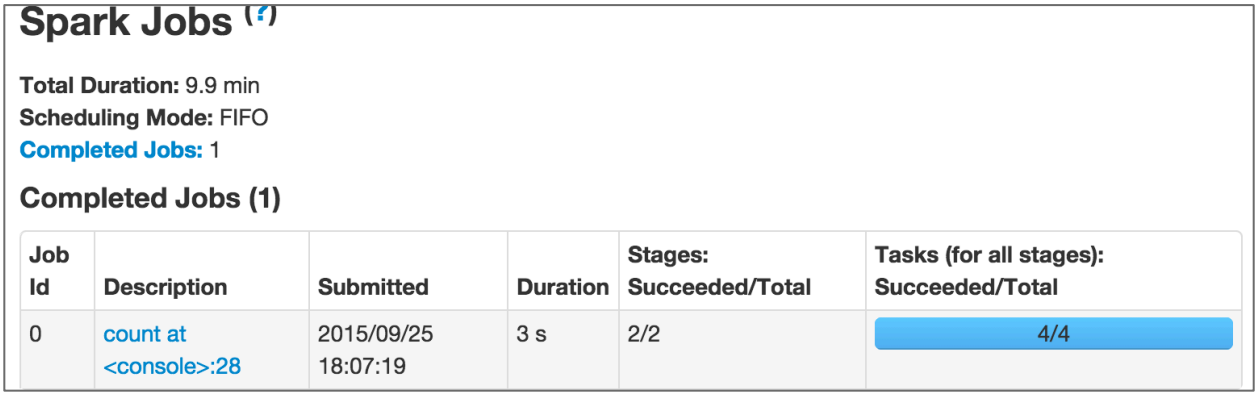
a. The last three lines here - represent the loading of the data using textFile and applying the `map` transformation to tokenize the line.

b. `|  ShuffledRDD[5] at distinct at <console>:25 []`

The `distinct` transformation results in a shuffle.

c. Usually there is a stage per RDD. However, there is no movement of data between child and parent till the `distinct` transformation. Thus all these RDDs are pipelined into one stage and the action will be in the second stage.

4. Spark Web UI



Number of stages =2

Job took 3 s

Click on the job in the Description column to go to the Job Details page as shown below.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	count at <console>:28 +details	2015/09/25 18:07:23	0.2 s	2/2			1053.0 B	
0	distinct at <console>:25 +details	2015/09/25 18:07:20	3 s	2/2	55.1 MB			1053.0 B

Stage 0 - distinct - took 3 s.

Stage 1 - count - took 0.2 s

Each stage has 2 tasks

6. After caching:

Spark Jobs (?)

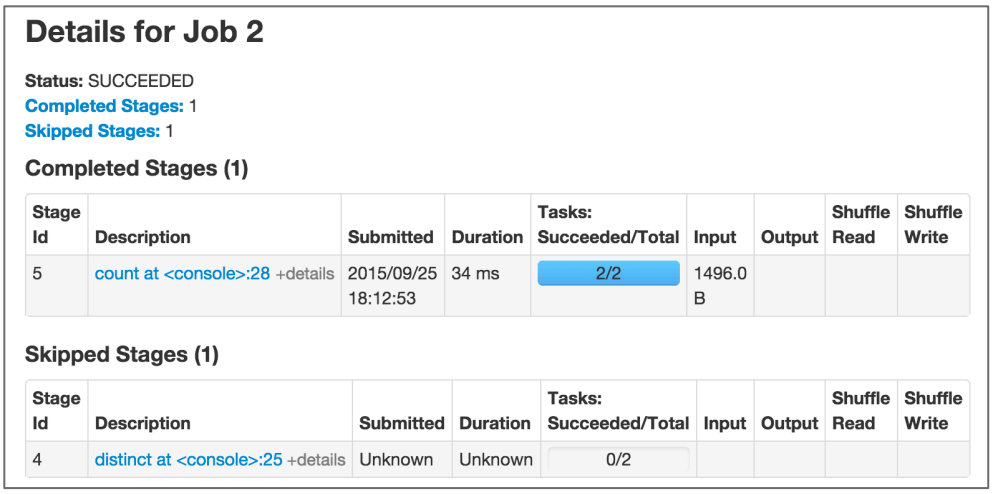
Total Duration: 18 min
Scheduling Mode: FIFO
Completed Jobs: 3

Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at <console>:28	2015/09/25 18:12:53	48 ms	1/1 (1 skipped)	2/2 (2 skipped)
1	count at <console>:28	2015/09/25 18:12:50	96 ms	1/1 (1 skipped)	2/2 (2 skipped)
0	count at <console>:28	2015/09/25 18:07:19	3 s	2/2	4/4

a. The first count (Job 0) took 3 s. The second count (Job 1) resulted in caching the RDD. The third count (Job 2) used the cached RDD and took only 48 ms.

b. There are two stages in this job and one is skipped.



Details for Job 2

Status: SUCCEEDED
Completed Stages: 1
Skipped Stages: 1

Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	count at <console>:28 +details	2015/09/25 18:12:53	34 ms	2/2	1496.0 B			

Skipped Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	distinct at <console>:25 +details	Unknown	Unknown	0/2				

c. Each stage has two tasks, but since the first stage (distinct) is skipped, there are no succeeded tasks for this stage.

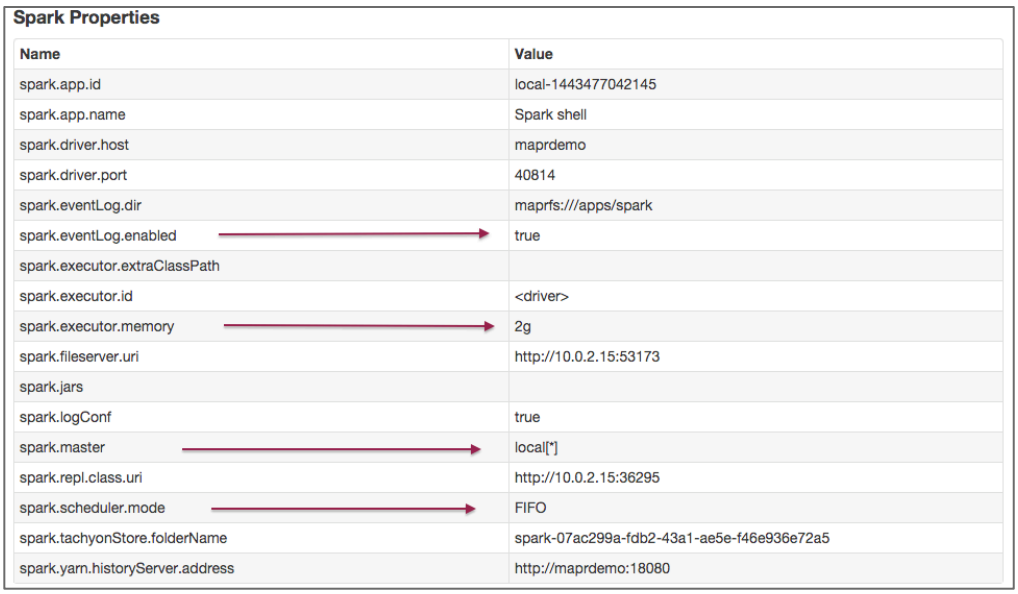
Try this! Explore the Spark Web UI.

Since you have cached the RDD, explore the Storage tab. How many partitions were cached? What is the size in memory? What fraction was cached?

Where would you view the thread dump?

Lab 6.2.1

1. Spark Web UI >> Environment tab



Spark Properties

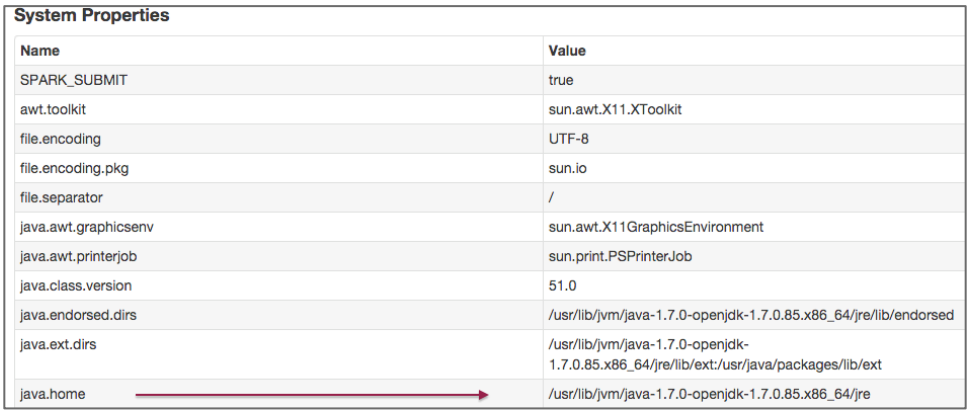
Name	Value
spark.app.id	local-1443477042145
spark.app.name	Spark shell
spark.driver.host	maprdemo
spark.driver.port	40814
spark.eventLog.dir	maprfs:///apps/spark
spark.eventLog.enabled	true
spark.executor.extraClassPath	
spark.executor.id	<driver>
spark.executor.memory	2g
spark.fileserver.uri	http://10.0.2.15:53173
spark.jars	
spark.logConf	true
spark.master	local[*]
spark.repl.class.uri	http://10.0.2.15:36295
spark.scheduler.mode	FIFO
spark.tachyonStore.folderName	spark-07ac299a-fdb2-43a1-ae5e-f46e936e72a5
spark.yarn.historyServer.address	http://maprdemo:18080

2. 2g

3. Yes. spark.eventLog.enabled

4. local[*]

5. FIFO



System Properties

Name	Value
SPARK_SUBMIT	true
awt.toolkit	sun.awt.X11.XToolkit
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.printerjob	sun.print.PSPrinterJob
java.class.version	51.0
java.endorsed.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.85.x86_64/jre/lib/endorsed
java.ext.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.85.x86_64/jre/lib/ext:/usr/java/packages/lib/ext
java.home	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.85.x86_64/jre

6. java.home

DEV 361 Appendix

Troubleshooting Common Errors

Note: If using Scala or Python, test using the Scala Interactive Shell or pyspark shell, respectively.

1. You use textFile method to load data from an external file into an RDD. You tested this in the Interactive shell. When you create a standalone application, you get an error. What are some possible causes?

   A: You may not have initialized SparkContext.

2. You have line of code that includes multiple steps such as loading the data, tokenizing it (map), filter, maps and distinct and you run into errors. A common error is of the form that the given RDD does not support a particular operation or that you cannot apply the operation to the array. What are some ways to debug this?

Note: In Scala, to see what type of RDD it is, type in RDD name in the Shell and enter.

To see what operations can be applied to the RDD, type in RDD name followed by a period + tab key.

For example:

```
val sunrdd =
sc.textFile("/path/to/file/sfpd.csv").map(x=>x.split(",").filter(
x=>x.contains("Sunday")).count

sunrdd.collect
```

Error message:

```
error: value collect is not a member of Long

sunrdd.collect
```

Is sunrdd an RDD? How can you find out what sunrdd is?

A: `sunrdd` ENTER

What can you do to your code?

A: Rather than writing all the operations in one step, break it down and use first() or take(n) to look at the data in each step. Also check what operations can be applied to the rdd by typing:

`sunrdd.` followed by TAB key

Note: A very common mistake is the path to the file. Spark does not check the path till an action is called. When testing in the shell, verify that the data did load by applying `first()` or `take(n)` before going any further.

Note: In an application, make sure that you import all the necessary packages; create a SparkContext and SQLContext (if this is needed).

Note: Handling missing dependencies in Jar file - refer to:

https://databricks.gitbooks.io/databricks-spark-knowledge-base/content/troubleshooting/missing_dependencies_in_jar_files.html

3. "Bad" Data: Inspect your data; understand your data.

    a. For example, you may have a csv file containing commas within a field value. If you just split on the comma, your data may not split as expected. Clean up the data or write a function to parse the data.

    b. Use filters where appropriate.

Null Pointer Exception in Scala

Scala provides a class called Option that can be used to handle null values.

Scala has a **null** value in order to communicate with Java, and should be used only for this purpose. In all other cases, use the Scala Option class. Instances of Option are either scala.Some or object None. Use Option with map, flatmap, filter or foreach.

For more information:

http://www.scala-lang.org/api/current/index.html#scala.Option

Beginning Scala: David Pollack

Scala Cookbook - Alvin Alexander

http://alvinalexander.com/scala/scala-null-values-option-uninitialized-variables