University of East London

Pioneering Futures Since 1898

RDD and DataFrame in Spark

Dr Amin Karami

a.karami@uel.ac.uk

www.aminkarami.com

CN7022 – Week 8
22 November 2019

Outline

- RDD Programming using Python
- RDD Disadvantages
- RDD vs DataFrame vs DataSet
- DataFrame Programming using SQL

University of East London

Learning Outcomes

- To understand the RDD Programming using Python
- To be able to explain the differences between RDD, DataFrame, and DataSet
- To be able to read data into Spark and complete the actions using RDD Python or DataFrame SQL.

The rest of term

- **Week 8 (today, 22nd Nov.):** RDD & DataFrame Programming
- **Week 9 (29th Nov.):** Probabilistic modelling using PySpark
- **Week 10 (6th Dec., CRWK preparation, <u>no class</u>)**
- **Week 11 (13th Dec., CRWK preparation, <u>no class</u>)**

<u>Additional booked rooms on December 2019:</u>

- **KD.2.14:**
  1. Monday (2/12, 9/12, 16/12): 10:00 – 13:00
  2. Wednesday (4/12, 11/12, 18/12): 10:00 – 13:00
- **KD.2.28E:**
  1. Monday (2/12, 9/12, 16/12): 10:00 – 17:00 (all day)
  2. Wednesday (4/12, 11/12, 18/12): 10:00 – 17:00 (all day)

University of East London

The Presentation (40% weighting)

- **Week 12 (Tuesday, 17th Dec.):** CRWK presentation
  - KD.2.14 and KD.2.15 [2pm-6pm]

- **Week 12 (Wednesday, 18th Dec.):** CRWK presentation
  - KD.2.14 and KD.2.15 [9am-1pm]

	Dec 17 TUE 2:00 PM 2:20 PM	Dec 17 TUE 2:20 PM 2:40 PM	Dec 17 TUE 2:40 PM 3:00 PM	Dec 17 TUE 3:00 PM 3:20 PM	Dec 17 TUE 3:20 PM 3:40 PM	Dec 17 TUE 3:40 PM 4:00 PM	Dec 17 TUE 4:00 PM 4:20 PM	Dec 17 TUE 4:20 PM 4:40 PM	Dec 17 TUE 4:40 PM 5:00 PM	Dec 17 TUE 5:00 PM 5:20 PM	Dec 17 TUE 5:20 PM 5:40 PM	Dec 17 TUE 5:40 PM 6:00 PM
0 participants	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3
Group 100												

Dec 18 WED 9:00 AM 9:20 AM	Dec 18 WED 9:20 AM 9:40 AM	Dec 18 WED 9:40 AM 10:00 AM	Dec 18 WED 10:00 AM 10:20 AM	Dec 18 WED 10:20 AM 10:40 AM	Dec 18 WED 10:40 AM 11:00 AM	Dec 18 WED 11:00 AM 11:20 AM	Dec 18 WED 11:20 AM 11:40 AM	Dec 18 WED 11:40 AM 12:00 PM	Dec 18 WED 12:00 PM 12:20 PM	Dec 18 WED 12:20 PM 12:40 PM	Dec 18 WED 12:40 PM 1:00 PM
✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3	✓0/3

- Book your presentation(**deadline: 10th Dec.**): https://bit.ly/2NX7bJq
- The participant should be your **group ID**.

University of East London

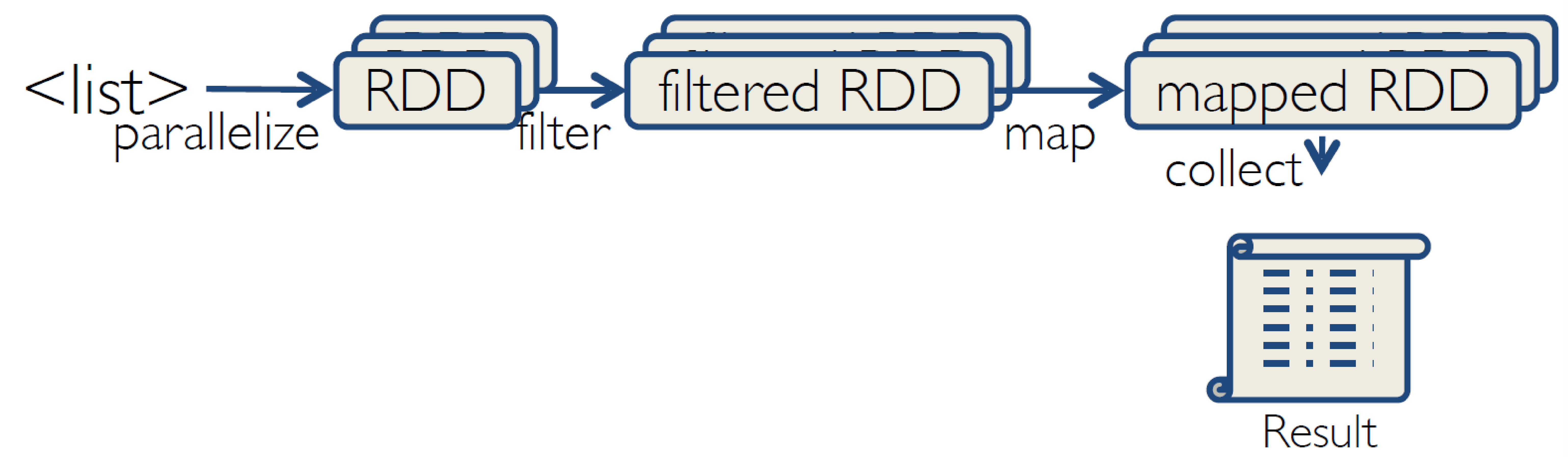
Launch PySpark

```
uel@uel-Deskop-VM: /usr/local/spark
uel@uel-Deskop-VM:~$ cd $SPARK_HOME
uel@uel-Deskop-VM:/usr/local/spark$ ./sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs
/spark-uel-org.apache.spark.deploy.master.Master-1-uel-Deskop-VM.out
failed to launch: nice -n 0 /usr/local/spark/bin/spark-class org.apache.spark.de
ploy.master.Master --host uel-Deskop-VM --port 7077 --webui-port 8080
full log in /usr/local/spark/logs/spark-uel-org.apache.spark.deploy.master.Maste
r-1-uel-Deskop-VM.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local
/spark/logs/spark-uel-org.apache.spark.deploy.worker.Worker-1-uel-Deskop-VM.out
localhost: failed to launch: nice -n 0 /usr/local/spark/bin/spark-class org.apac
he.spark.deploy.worker.Worker --webui-port 8081 spark://uel-Deskop-VM:7077
localhost: full log in /usr/local/spark/logs/spark-uel-org.apache.spark.deploy.w
orker.Worker-1-uel-Deskop-VM.out
uel@uel-Deskop-VM:/usr/local/spark$ pyspark
```

Tip: if you want to load/save data from/into HDFS, you need run Hadoop engine as well by `start-all.sh`

Working with Spark

1. Create a RDD from a data source (Create `<list>`)
2. Apply transformations to a RDD (e.g., `map, filter`)
3. Apply actions to a RDD: (e.g., `collect, count`)



University of East London

(step 1) data loading: Create RDD

- We start creating a data with `parallelize()` function.
- No computation occurs with `sc.parallelize()` and Spark only records how to create the RDD with **four partitions**.

```
data = [1,2,4,7,11,15,20]
```

```
data
```

```
[1, 2, 4, 7, 11, 15, 20]
```

```
rdd = sc.parallelize(data,4)
```

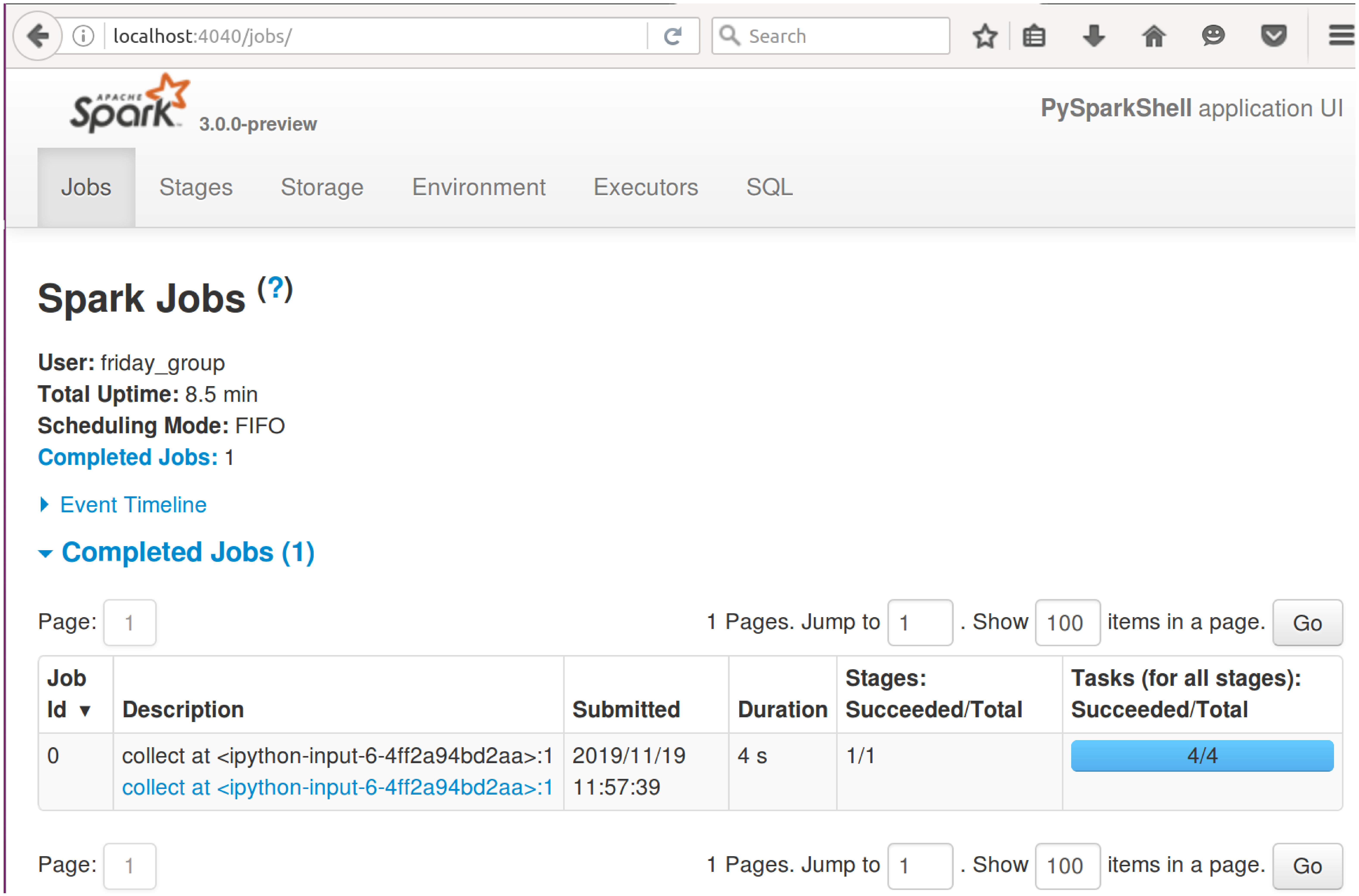
```
rdd
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:247
```

```
rdd.collect()
```

```
[1, 2, 4, 7, 11, 15, 20]
```

PySpark Shell: localhost:4040



Create RDD from a file

- We can read data from HDFS, text, Amazon S3, Apache Hbase, and many more.

```
In [3]: file1 = sc.textFile("/home/uel/textfile", 3)
        file1

Out[3]: /home/uel/textfile MapPartitionsRDD[2] at textFile at NativeMethodAccessorImpl.java:0
```

- RDD distribute the data into 3 partitions
- Again, this is **Lazy evaluation**: no execution happens now

(step 2) Spark: Transformation

Transformation	Description
`map(func)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(func)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([numTasks]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(func)`	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)

Transformation example

```
rdd.collect()
```

```
[1, 2, 4, 7, 11, 15, 20]
```

```
rdd1=rdd.map(lambda x:(x+2)*4)
rdd1.collect()
```

```
[12, 16, 24, 36, 52, 68, 88]
```

```
rdd2=rdd.filter(lambda x:(x+2)*4)
rdd2.collect()
```

```
[1, 2, 4, 7, 11, 15, 20]
```

```
rdd3=rdd.filter(lambda x : x % 3==0)
rdd3.collect()
```

```
[15]
```

```
rdd4=rdd.map(lambda x : x % 3==0)
rdd4.collect()
```

```
[False, False, False, False, False, True, False]
```

```
rdd5 = sc.parallelize([4,2,2,6,7,7,19,40,41,40,40])
```

```
rdd5.distinct()
```

```
PythonRDD[15] at RDD at PythonRDD.scala:53
```

```
rdd5.distinct().collect()
```

```
[4, 40, 41, 2, 6, 7, 19]
```

Transformation example

```
rdd6=sc.parallelize([1,2,3,4])
```

```
rdd7=rdd6.map(lambda x : [x,x+2,x+7])
rdd7.collect()
```

```
[[1, 3, 8], [2, 4, 9], [3, 5, 10], [4, 6, 11]]
```

```
rdd8=rdd6.flatMap(lambda x : [x,x+2,x+7])
rdd8.collect()
```

```
[1, 3, 8, 2, 4, 9, 3, 5, 10, 4, 6, 11]
```

- The difference of `map` and `flatMap`.
- If you want to have map-reduce programing, you need `flatMap`. Because `map` returns list, but `flatMap` returns a sequence of values.

(step 3) Spark: Action

Action	Description
`reduce(`*func*`)`	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
`take(`*n*`)`	return an array with the first *n* elements
`collect()`	return all the elements as an array WARNING: make sure will fit in driver program
`takeOrdered(`*n*`, `*key=func*`)`	return n elements ordered in ascending order or as specified by the optional key function

University of East London

Action example

```
print (rdd7.collect())
print (rdd8.collect())
```

```
[[1, 3, 8], [2, 4, 9], [3, 5, 10], [4, 6, 11]]
[1, 3, 8, 2, 4, 9, 3, 5, 10, 4, 6, 11]
```

```
rdd8.reduce(lambda a,b:a*b)
```

```
68428800
```

```
rdd8.collect()
```

```
[1, 3, 8, 2, 4, 9, 3, 5, 10, 4, 6, 11]
```

```
rdd8.take(4)
```

```
[1, 3, 8, 2]
```

```
rdd8.takeOrdered(4)
```

```
[1, 2, 3, 3]
```

Action (key-value RDDs) example

Key-Value Transformation	Description
`reduceByKey(func)`	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
`sortByKey()`	return a new dataset (K,V) pairs sorted by keys in ascending order
`groupByKey()`	return a new dataset of (K, Iterable<V>) pairs

```
In [35]: rdd=sc.parallelize([(1,2),(4,5),(1,7),(4,2),(5,3),(4,9)])
         rdd.collect()

Out[35]: [(1, 2), (4, 5), (1, 7), (4, 2), (5, 3), (4, 9)]

In [36]: rdd=rdd.reduceByKey(lambda a,b: a + b)
         rdd.collect()

Out[36]: [(4, 16), (1, 9), (5, 3)]

In [37]: rdd2=sc.parallelize([(1,'a'),(4,'c'),(4,'c'),(2,'b'),(1,'d')])

In [38]: rdd3=rdd2.sortByKey()
         rdd3.collect()

Out[38]: [(1, 'a'), (1, 'd'), (2, 'b'), (4, 'c'), (4, 'c')]
```

University of East London

RDD vs DataFrame

- **RDD**: The RDD APIs have been on Spark in 1.0 release. It is a distributed collection of data elements spread across many machines in the cluster. RDDs are a set of Java or Scala objects representing data.
- **DataFrames**: Spark introduced DataFrames in Spark 1.3 release. A DataFrame is a distributed collection of data organized into named columns, like a table in a relational database, using off-heap storage.
- **DataSet**: Spark introduced Dataset in Spark 1.6 release. It is an extension of DataFrame API that provides the benefits of the Catalyst query optimizer and off heap storage mechanism.

University of East London

RDD disadvantages

1. **Outdated:** DataFrame and Dataset are distributed collections of data with the benefits of Spark SQL's optimized execution engine.

2. **Hard to Use:** RDD needs Python/Scala/Java coding, but DataFrame and Dataset need SQL-like queries, and anyone who knows SQL will understand it in one go.
3. **Slow Speed:** the main reason to not use RDD is its performance, which can be a major issue for some applications.

DataFrame example

- We need to convert RDD/Hive to DataFrame, or find a relational dataset like csv, SQL, etc.
- Download data (FIFA 18 Player Dataset) from https://www.kaggle.com/thec03u5/fifa-18-demo-player-dataset or from here. We are going to work with *CompleteDataset.csv* file.
- Tip1: A `spark.read.csv` package converts `*.csv` files to `DF`. If your data format is different, then you need to use another package, such as `SQLContext` or `pyspark.sql` to create DF.
- Tip2: Usually we do not use lambda format in DataFrame, instead, we use SQL-like commands.

University of East London

(1) Read data from CSV

```
fifa_df = spark.read.csv("/home/friday_group/Desktop/CompleteDataset.csv",
                         inferSchema=True, header=True)
```

- Or:

```
fifa_df2 = sqlContext.read.load('/home/friday_group/Desktop/CompleteDataset.csv',
                                format = 'com.databricks.spark.csv',
                                header = 'true', inferSchema = 'true')
```

(2) The Schema of DF

To have a look at the structure of the Dataframe, we'll use the `printSchema()` method.

```
fifa_df.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Photo: string (nullable = true)
 |-- Nationality: string (nullable = true)
 |-- Flag: string (nullable = true)
 |-- Overall: integer (nullable = true)
 |-- Potential: integer (nullable = true)
 |-- Club: string (nullable = true)
 |-- Club Logo: string (nullable = true)
 |-- Value: string (nullable = true)
 |-- Wage: string (nullable = true)
 |-- Special: integer (nullable = true)
 |-- Acceleration: string (nullable = true)
 |-- Aggression: string (nullable = true)
 |-- Agility: string (nullable = true)
 |-- Balance: string (nullable = true)
 |-- Ball control: string (nullable = true)
 |-- Composure: string (nullable = true)
 |-- Crossing: string (nullable = true)
 |-- Curve: string (nullable = true)
 |-- Dribbling: string (nullable = true)
```

(3) Columns information in DF

```
fifa_df.columns
```

```
['_c0',
 'Name',
 'Age',
 'Photo',
 'Nationality',
 'Flag',
 'Overall',
 'Potential',
 'Club',
```

```
fifa_df.count()
```

```
17981
```

```
len(fifa_df.columns)
```

```
75
```

University of East London

(4) Select in DF

```
fifa_df.select('Name','Nationality','club').show()
```

```
+-----------------+-----------+-------------------+
|             Name|Nationality|               club|
+-----------------+-----------+-------------------+
|Cristiano Ronaldo|   Portugal|     Real Madrid CF|
|         L. Messi|  Argentina|       FC Barcelona|
|           Neymar|     Brazil|Paris Saint-Germain|
|        L. Suárez|    Uruguay|       FC Barcelona|
|         M. Neuer|    Germany|   FC Bayern Munich|
|   R. Lewandowski|     Poland|   FC Bayern Munich|
|           De Gea|      Spain|  Manchester United|
|        E. Hazard|    Belgium|            Chelsea|
|         T. Kroos|    Germany|     Real Madrid CF|
|       G. Higuaín|  Argentina|           Juventus|
|     Sergio Ramos|      Spain|     Real Madrid CF|
|     K. De Bruyne|    Belgium|    Manchester City|
|      T. Courtois|    Belgium|            Chelsea|
|       A. Sánchez|      Chile|            Arsenal|
|        L. Modrić|    Croatia|     Real Madrid CF|
|          G. Bale|      Wales|     Real Madrid CF|
|        S. Agüero|  Argentina|    Manchester City|
|     G. Chiellini|      Italy|           Juventus|
|        G. Buffon|      Italy|           Juventus|
|        P. Dybala|  Argentina|           Juventus|
+-----------------+-----------+-------------------+
only showing top 20 rows
```

```
fifa_df.select('Name','Long shots').distinct().show()
```

```
+-----------------+----------+
|             Name|Long shots|
+-----------------+----------+
|Cristiano Ronaldo|        92|
|      J. Cuadrado|        80|
|      M. Brozović|        79|
|          A. Rami|        58|
|       D. Abraham|        65|
|     Borja Bastón|        73|
|       J. Montero|        68|
|      T. Barnetta|        74|
|          Wallace|        26|
|       A. Barreca|        42|
|     Y. Benalouane|       39|
|          Juankar|        64|
|        D. Appiah|        38|
|   Rafael Martins|        69|
|          Granell|        77|
|     A. Cornelius|        68|
|         J. Henry|        75|
|        M. Ozdoev|        69|
|            Fábio|        58|
|       T. Dingomé|        60|
+-----------------+----------+
only showing top 20 rows
```

University of East London

(4) GroupBy in DF

```
fifa_df.groupBy("age").count().show()
```

```
+---+-----+
|age|count|
+---+-----+
| 29| 1121|
| 30|  804|
| 34|  272|
| 28| 1051|
| 22| 1324|
| 35|  191|
| 16|   13|
| 47|    1|
| 43|    2|
| 31|  671|
| 18|  672|
| 27| 1152|
| 17|  258|
| 26| 1202|
| 19| 1069|
| 23| 1394|
| 41|    3|
| 38|   36|
| 40|    8|
| 25| 1522|
+---+-----+
only showing top 20 rows
```

(5) SQL on DF

```
fifa_df_sql = sqlContext.read.csv("/home/friday_group/Desktop/CompleteDataset.csv", header= True)
```

```
# Register the DataFrame as a SQL temporary view
fifa_df_sql.createOrReplaceGlobalTempView("fifa")
```

```
sqlDF = spark.sql("SELECT age, count(*) as count FROM global_temp.fifa GROUP BY age")
```

```
sqlDF.show()
```

```
+---+-----+
|age|count|
+---+-----+
| 29| 1121|
| 30|  804|
| 34|  272|
| 28| 1051|
| 22| 1324|
| 35|  191|
| 16|   13|
| 47|    1|
| 43|    2|
| 31|  671|
| 18|  672|
| 27| 1152|
| 17|  258|
| 26| 1202|
| 19| 1069|
| 23| 1394|
| 41|    3|
| 38|   36|
| 40|    8|
| 25| 1522|
+---+-----+
only showing top 20 rows
```

Spark SQL and DataFrames:
https://spark.apache.org/docs/2.3.0/sql-programming-guide.html

Summary

- Introduced RDD Programming
- Discussed RDD and DataFrame
- Exercised RDD programming using Python
- Exercised DataFrame programming using SQL