Lab 3.1 : RDD Basics operations

Overview

In this lab, we will work with RDDs at a basic level, including: Learn

- basic operations like filter / map / count
- Work with larger sized RDDs
- Load multiple files into a single RDD Save
- computed RDDs

Note : zeppelin means use speak shell later we will use zeppelin.

Builds on

Lab 2.2: Spark Shell

Run time

30-40 minutes

Load a small file, then filter it

We supply a number of text files of varying sizes in *data/twinkle* including (inside spark labs):

- *sample.txt*: A few lines
- *1M.data*, *10M.data*: 1MB, 10MB text file
- *100M.data.zip*, *200M*, *500M*, *1G*: Various size files zipped up to save space We've
  - unzipped some of them (100M and 500M) for easy use.
  - If you want to unzip and use others, see notes at end of this lab.

Tasks

- Go to your spark shell.
- In the shell, open *sample.txt*, as shown below
  - The shell will respond with the type of the result

```
// Scala
> val f = sc.textFile("spark-labs/data/twinkle/sample.txt")
f: org.apache.spark.rdd.RDD[String] = spark-
labs/data/twinkle/sample.txt MapPartitionsRDD[458] at textFile at
<console>:24
```

```
# Python
> f = sc.textFile("spark-labs/data/twinkle/sample.txt")
```

Notes on Python Lambdas

Lambdas in Python have the syntax shown below. x is a formal parameter, and can be called anything. The example at bottom checks if the the string `"Hello World"` is contained in x.

```
lambda x: "function expression"
e.g.
lambda x: "Hello World" in x
```

Tasks

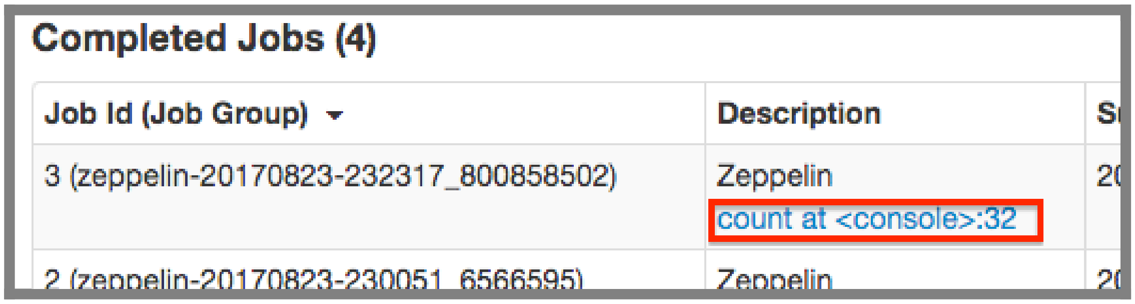
Let's find how many lines contain the word "twinkle". We'll do this using the `filter` function.

- Filter for lines containing "twinkle" Count
- the number of lines returned
    - You can use `count()` on the filtered RDD
    - You should see a count of the number of lines containing twinkle

```
// Scala
> val filtered = f.filter(line => line.contains("twinkle"))
// How do you count?
```

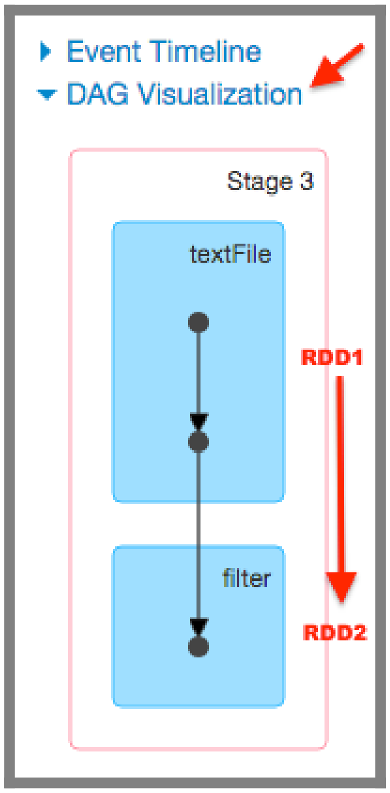
```
# Python
> filtered = f.filter(lambda line: "twinkle" in line)
// How do you count?
```

- Goto Spark shell UI (as done previously) Inspect
- the 'Stages' section in the UI
- Note how the above transformations are executed (should be the first stage listed in the table of "**Completed Stages**"
- Next, view the DAG visualiation in the Jobs tab at follows
    - Go to the Jobs tab - the above transformation should be the first job listed Click on
    - the link in the **Description**, as shown below



Job Id (Job Group) ▾	Description	S
3 (zeppelin-20170823-232317_800858502)	Zeppelin count at <console>:32	20
2 (zeppelin-20170823-230051_6566595)	Zeppelin	20

- Expand the **DAG Visuaization** node to view it, as shown below



We can observe some of this behavior in the shell by changing the logging

- In the shell, change the logging level to "INFO" as shown below (the default is "WARN")
  - Create the filtered RDD, then run the count again
  - You'll see a lot of logging output, but not until the count is executed (Spark is always lazy)
  - When done, set the level to "WARN" again

```
> sc.setLogLevel("INFO")
// Create the filtered RDD, run the count - review the logging for both
...
// Run all the above in spark shell

// Once done, set the log level back to WARN.
> sc.setLogLevel("WARN") // Set level back to WARN
```

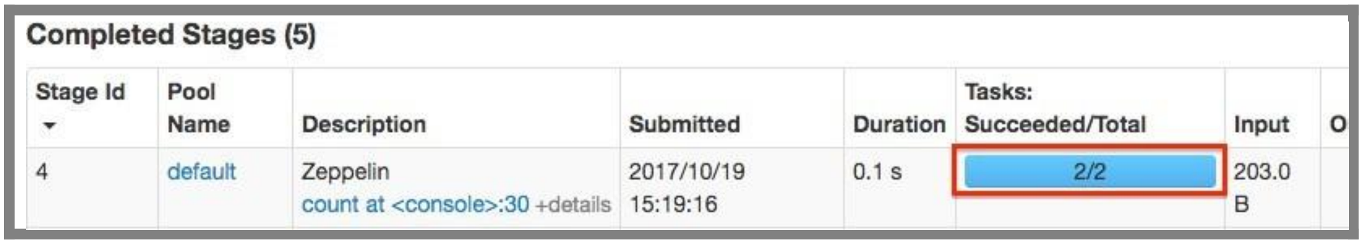
Process a large file

We'll do some processing now on much larger files, and then view the processing in the UI

NOTE: All data files are in the *spark-labs/data/twinkle* folder

Tasks

- In the Spark Shell, load the *100M.data* file (it is about 100MB in size).
  - Use `sc.textFile(" ....")`
- Count the number of lines that have the word "diamond" Use
  - `filter` and `count`
- Check how many 'tasks' are used in the above calculation Check
  - **Stages** tab in the spark shell UI - we illustrate below



Completed Stages (5)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	O
4	default	Zeppelin count at <console>:30 +details	2017/10/19 15:19:16	0.1 s	2/2	203.0 B	

- Count the number of lines that do NOT have the word 'diamond'
  - **Scala:** Use the negative operator `!`
  - **Python:** Use the negative operator `not`
- Verify that both counts add up to the total line count How
  - do you get the total line count?
- Try the above with larger data files : 500M.data and 1G.data
  - Note the time taken and the number of tasks.
  - You can also look at the executor's stderr output, which is available from the

stage detail in the UI

        - Click the stage link (`count` in this example) in the Description to get to the stage detail

    - You can also look at the UI for the Spark master
        - See instructions in earlier lab on accessing this.
        - In this UI, if you drill down to a worker, you can view its stout/stderr

Loading multiple files

You can use wildcards in the filename to specify multiple files, e.g. ***.data**

Tasks

- Load some of the files like this and then filter for lines containing the word "diamond"
- Count them, and review the UI statistics

Writing data

You can easily write an RDD to disk using `saveAsTextFile("filename")` as shown below

Tasks

- Save the filtered data from above, as shown

```
> filtered.saveAsTextFile("spark-labs/data/twinkle/out1")
```

- View the data that was written on the filesystem (e.g. through a file explorer) Note
    - how the data is in a folder named OUT1.
    - There are many data files within OUT1.
    - Why? Because each partition is written separately. Think about why this is.

If Spark wanted to write all the partitioned data to a single file, the data would have to be repartitioned (or coalesced - e.g. `filtered.coalesce(1)...`) so that this could be done. This would require that a single worker hold all the data in memory at once. This is not a good practice. If you need a single file, it's not di"cult to create it (e.g. via HDFS tools if using HDFS).

- **[Optional]** : Try writing the filtered RDD as a single file

- Does it succeed? If so, how long does it take?

Notes: Unzipping Other Twinkle Files (if needed)

To use any of the other large twinkle data files, you need to unzip them on the local file system then copy them to HDFS, as illustrated below for 1G.data

Tasks

- Open a terminal prompt in the guacamole window, and execute the following.

```
# Go to local twinkle folder
$ cd /spark-labs/data/twinkle
# Unzip the local file
$ unzip local 1G.data file
# Copy unzipped local file to HDFS (enter command all on one line)
$ hdfs dfs -copyFromLocal /spark-labs/data/twinkle/1G.data
       /user/zeppelin/spark-labs/data/twinkle
```

Summary

You've done some straight forward transformations, and worked with various data files. We will work with more complex Spark processing capabilities, soon.