

Aims

This exercise aims to get you to:

- Analyze data using Spark shell
- Monitor Spark tasks using Web UI

Background

Spark is already installed on the virtual machine image. Please follow the instructions to do the installation and configuration of Spark in the specified folder.

The detailed Spark programming guide is available at:

<http://spark.apache.org/docs/latest/programming-guide.html>

The transformation and action functions examples are available at:

<http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html>

A tutorial of Scala is available at:

http://docs.scala-lang.org/tutorials/?_ga=1.99469143.850382266.1473265612

The answers to the questions are given at the end of this file. Please try to answer all questions by yourself utilizing the above documents, and then check your results with the answers provided.

Install Spark

1. Install openjdk 8

The installation of spark-2.3.0 requires openjdk 8, which is not installed in the virtual machine. You can install it by:

```
$ sudo apt-get install openjdk-8-jdk
```

If the command cannot be successfully completed, you need to add the openjdk 8 source, and then finish the installation:

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
```

This may take several minutes depending on the network.

You also need to update JAVA_HOME. In ~/.bashrc, edit:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

2. Create a working directory for Spark

```
$ mkdir ~/workdir
```

Then get into the directory:

```
$ cd ~/workdir
```

Download the Spark package by the command:

```
$ wget http://apache.melbourneitmirror.net/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz
```

Then unpack the package:

```
$ tar xvf spark-2.3.0-bin-hadoop2.7.tgz
$ mv spark-2.3.0-bin-hadoop2.7 spark
```

Now you have Spark installed under ~/workdir/spark. We need to configure this folder as the working directory of Spark.

Open the file ~/.bashrc and add the following lines to the **end** of this file:

```
export SPARK_HOME=/home/comp9313/workdir/spark
export PATH=$SPARK_HOME/bin:$PATH
```

Save the file, and then run the following command to take these configurations into effect:

```
$ source ~/.bashrc
```

Next, start HDFS by:

```
$ start-dfs.sh
```

Use the following command for opening Spark shell:

```
$ spark-shell
```

Interactive Analysis with the Spark Shell

1. Load and inspect data from a text file:

1. Create an **RDD** from local files using textFile()

rdd是spark的灵魂，中文翻译弹性分布式数据集，一个rdd代表一个可以被分区的只读数据集。rdd内部可以有許多分区(partitions)，每个分区又拥有大量的记录(records)

```
$ scala> val textFile =  
sc.textFile("file:///home/comp9313/workdir/spark/README.md")
```

Spark's primary abstraction is a distributed collection of items called a **Resilient Distributed Dataset (RDD)**. RDDs can be **created from Hadoop InputFormats (such as HDFS files)** or by transforming other RDDs. This command makes a new RDD from the text of the README file in the Spark source directory.

You can apply the RDD transformation and action functions on “textFile”.

2. **Count the number of items in an RDD (count() is an action)**

Definition: `def count(): Long`

```
$ scala> textFile.count()
```

You should see results: “res0: Long = 103”

3. Get the first item in an RDD

Definition: `def first(): T`

```
$ scala> textFile.first()
```

You should see results: “res1: String = # Apache Spark”

4. Get lines containing “Spark” using the function `filter()`

Definition: `def filter(f: T => Boolean): RDD[T]`

```
$ scala> val linesWithSpark = textFile.filter(line =>  
line.contains("Spark"))
```

You can also use underscore in the argument, that is, `filter(_.contains("Spark"))`. Try to count the items in `linesWithSpark`.

5. Use the function `collect()` to see the contents of `linesWithSpark`

Definition: `def collect(): Array[T]`

```
$ scala> linesWithSpark.collect()
```

```
res22: Array[String] = Array(# Apache Spark, Spark is a fast and general cluster
computing system for Big Data. It provides, rich set of higher-level tools incl
uding Spark SQL for SQL and DataFrames,, and Spark Streaming for stream processi
ng., You can find the latest Spark documentation, including a programming, ## Bu
ilding Spark, Spark is built using [Apache Maven](http://maven.apache.org/)., To
build Spark and its example programs, run:, You can build Spark using more than
one thread by using the -T option with Maven, see ["Parallel builds in Maven 3"
](https://cwiki.apache.org/confluence/display/MAVEN/Parallel+builds+in+Maven+3).
, ["Building Spark"](http://spark.apache.org/docs/latest/building-spark.html).,
For developing Spark using an IDE, see [Eclipse](https://cwiki.apache.org/con..
```

6. Print all the items in linesWithSpark

Definition: `def foreach(f: T => Unit)`

```
$ scala> linesWithSpark.foreach(println)
```

`println()` is a function, and it is used as an argument in function `foreach()`.

7. Use function `map()` to map each line to the number of words contained in it

Definition: `def map[U: ClassTag](f: T => U): RDD[U]`

```
$ scala> val lineNumOfWords = textFile.map(line => line.split("
").size)
```

The argument of `map()` is an anonymous function, which takes a line as the input, and returns the number of words (separated by space). Check the contents of `lineNumOfWords`.

8. Find the largest number of words contained in a line using `reduce()`

Definition: `def reduce(f: (T, T) => T): T`

```
$ scala> lineNumOfWords.reduce( (a, b) => if (a > b) a else b)
```

You should see the result is 22. The `reduce` function takes an anonymous function as an argument, which takes two arguments and returns the larger one.

You can also call functions declared elsewhere. For example, you can use `Math.max()` function to make this code easier to understand:

```
$ scala> import java.lang.Math
```

```
$ scala> lineNumOfWords.reduce( (a, b) => Math.max(a, b) )
```

9. Convert RDD `textFile` to an array of words using `flatMap()`

Definition: `def flatMap[U: ClassTag](f: T => TraversableOnce[U]): RDD[U]`

```
$ scala> val words = textFile.flatMap(_.split(" "))
```

This will split each line to a list of words, and store all of them in one array. You can compare the result obtained by flatMap() with that obtained by map(). **What are the differences?**

```
$ scala> val words2 = textFile.map(_.split(" "))
```

10. Count the distinct words in textFile using distinct()

Definition: `def distinct(): RDD[T]`

```
$ scala> words.distinct().count()
```

Compare the results with words.count(). You can ignore “()” if there is no argument, that is, words.distinct.count.

11. **(Question)** Find the longest line together with the length in textFile.

Hint: first map a line to a pair of (line, length), and then use reduce() to find the longest line. To access the second field of an argument x, you can use x._2.

12. **(Question)** Print the lines containing “Spark” with line numbers (starting from 0). Each line is printed in format of:

Line Number in textFile: the contents of the line

Hint: Use function zipWithIndex().

Definition: `def zipWithIndex(): RDD[(T, Long)]`

Zip the elements of the RDD with its element indexes. The indexes start from 0.

2. More operations on pair RDD:

1. Download the data set auctiondata.csv from the course webpage.

Define the mapping for the input variables. They are used to refer to different fields of the data set.

```
$ scala> val aucid = 0
$ scala> val bid = 1
$ scala> val bidtime = 2
```

```
$ scala> val bidder = 3
$ scala> val bidderrate = 4
$ scala> val openbid = 5
$ scala> val price = 6
$ scala> val itemtype = 7
$ scala> val dtl = 8
```

2. Load data into Spark

```
$ scala> val auctionRDD =
sc.textFile("file:///home/comp9313/auctiondata.csv").map(_.split(","))
```

In auctionRDD, each item is an array containing 9 fields, and you can use the defined variables to access each field.

3. Count the total number of item types that were auctioned.

```
$ scala> auctionRDD.map(_(itemtype)).distinct.count,
```

or

```
$ scala> auctionRDD.map(x => x(itemtype)).distinct.count
```

Each item in auctionRDD is an array of String objects. `x(itemtype)` is equivalent to `x(7)`, and it is used to get the 8th object in the array. You can also use `x._7` to do the same work.

4. (Question) What is the total number of bids per item type? The output is a list of key-value pairs <item type, number of bids>.

Hint: First create a pair RDD by mapping each record to a pair of (item type, 1), and then use `reduceByKey()` to do the aggregation for each item type

Definition: `def reduceByKey(func: (V, V) => V): RDD[(K, V)]`

5. (Question) Across all auctioned items, what is the maximum number of bids?

Hint: First use `reduceByKey()` to count the number of bids for each auctioned item, and then find the maximum number using `reduce()`

6. (Question) Across all auctioned items, what are the top-5 items that have the most number of bids?

Hint: First use `reduceByKey()` to count the number of bids for each auctioned item, and then use `sortByKey(false)` to sort the key-value pairs in descending order. Note that `sortByKey()` works on keys, not values, and thus you need to swap the key and value. You can do this by “`map(x => (x._2, x._1))`” or “`map(x => x.swap)`”. Finally (use `take()` to get the top-5 results, and swap the key and value back.

Definition: `def sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.size): RDD[P]`

3. Do word count in Spark shell

Start HDFS, get the file “pg100.txt” from WebCMS3 and put it to HDFS

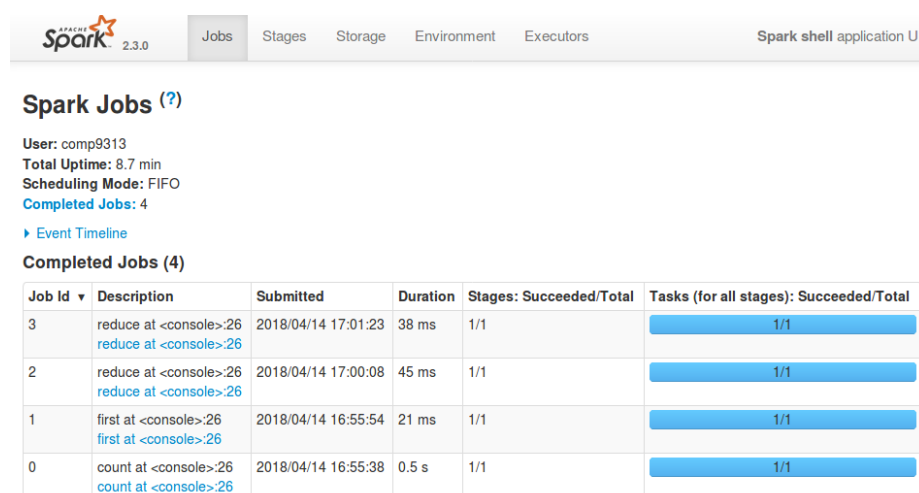
```
$ hdfs dfs -put pg100.txt
```

Load the file into Spark from HDFS, and use the functions `map()`, `flatMap()`, `reduceByKey()` to do word count (split the documents by the space character). Finally, store the results in **HDFS using `saveAsTextFile()`** and check the output.

RDD partitions: In the function `reduceByKey()`, set the number of tasks to 3, and check the results again (each task will be processed by one reducer, and thus three output files) .

Spark Web UI

Browse the web interface for the information of Spark Jobs, storage, etc. at: **<http://localhost:4040>**. You will see something like:



Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	reduce at <console>:26 reduce at <console>:26	2018/04/14 17:01:23	38 ms	1/1	1/1
2	reduce at <console>:26 reduce at <console>:26	2018/04/14 17:00:08	45 ms	1/1	1/1
1	first at <console>:26 first at <console>:26	2018/04/14 16:55:54	21 ms	1/1	1/1
0	count at <console>:26 count at <console>:26	2018/04/14 16:55:38	0.5 s	1/1	1/1

You can click each task to see more details of the execution.

Answers:

Note that for all answers you can define some intermediate variables to make the command more clear.

Interactive Analysis with the Spark Shell

```
1.11. $ scala> textFile.map(line => (line, line.length)).reduce((a, b) => if(a._2 > b._2) a else b)
```

Result:

```
res65: (String, Int) = (You can build Spark using more than one thread by using the -T option with Maven, see ["Parallel builds in Maven 3"](https://cwiki.apache.org/confluence/display/MAVEN/Parallel+builds+in+Maven+3),195)
```

```
1.12. $ scala> textFile.zipWithIndex().filter(a => a._1.contains("Spark")).foreach(a => println(a._2.toString() + ": " + a._1))
```

```
2.4. $ scala>
```

```
auctionRDD.map(x=>(x.itemtype),1)).reduceByKey(_+_).collect()
```

```
Or $ scala> auctionRDD.map(x=>(x.itemtype),1)).reduceByKey((a, b) => a + b).collect()
```

Result:

```
res66: Array[(String, Int)] = Array((xbox,2784), (palm,5917), (cartier,1953))
```

```
2.5. $ scala>
```

```
auctionRDD.map(x=>(x.aucid),1)).reduceByKey(_+_).reduce((a, b) => if(a._2 > b._2) a else b)._2
```

Result: 75

```
2.6. $ scala> auctionRDD.map(x=>(x.aucid),1)).reduceByKey(_+_).map(x => (x._2, x._1)).sortByKey(false).take(5).map(x => (x._2, x._1))
```

or

```
$ scala>
```

```
auctionRDD.map(x=>(x.aucid),1)).reduceByKey(_+_).map(_._swap).sortByKey(false).take(5).map(_._swap)
```

Result:

```
res: Array[(String, Int)] = Array((8214355679,75), (8212629520,57), (3023174478,54), (3020532816,51), (8212602164,50))
```

```
3. $ scala> val textFile = sc.textFile("pg100.txt")
```

```
$ scala> textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b).saveAsTextFile("output")
```

```
$ scala> textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b, 3).saveAsTextFile("output")
```