Hands-On Exercises

Hands-On Exercise: Viewing the Spark Documentation

In this Exercise you will familiarize yourself with the Spark documentation.

1. **V**isit the Spark documentation https://spark.apache.org/docs/<version number>/
2. From the **Programming Guides** menu, select the **Spark Programming Guide**. Briefly review the guide. You may wish to bookmark the page for later review.
3. From the **API Docs** menu, select either **Scaladoc** or **Python API**, depending on your language preference. Bookmark the API page for use during class. Later exercises will refer you to this documentation.

Hands-On Exercise: Using the Spark Shell

- **In this Exercise you will start the Spark Shell and verify the presence of your Spark Context.**
- You may choose to do this exercise using either Scala or Python. Follow the Instructions below for Python, or skip to the next section for Scala.
- Most of the later exercises assume you are using Python, but Scala solutions are provided , so you should feel free to use Scala if you prefer.

Using the Python Spark Shell

1. In a terminal window, start the pyspark shell:

```
$ pyspark
```

2. You may get several INFO and WARNING messages, which you can disregard. If you don t see the In[*n*]> prompt after a few seconds, hit Return a few times to clear the screen output.

3. Spark creates a SparkContext object for you called sc. Make sure the object exists:

```
pyspark> sc
```

Pyspark will display information about the sc object such as <pyspark.context.SparkContext at 0x2724490>

Using the Python Spark Shell

3. Context methods:
   type sc. (sc followed by a dot) and then the [TAB] key.
4. You can exit the shell by typing exit.

Using the Scala Spark Shell

1. **In a terminal window, start the Scala Spark Shell:**

```
$ spark-shell
```

   You may get several INFO and WARNING messages, which you can disregard. If you don t see the scala> prompt after a few seconds, hit Enter a few times to clear the screen output.

2. **Spark Creates a SparkContext object for you called sc. Make sure the object exists:**

```
scala>
```

   Scala will display information about the sc object such as:
   res0: org.apache.spark.SparkContext =
   org.apache.spark.SparkContext@2f0301fa

Using the Scala Spark Shell

3. Using command completion, you can see all the available SparkContext methods:
   type sc. (sc followed by a dot) and then the [TAB] key.
4. You can exit the shell by hitting Ctrl+c or by typing exit.

Hands-On Exercise: Getting Started with RDDs

LOAD AND VIEW TEXT FILE

1. Start the Spark Shell if you exited it from the previous exercise. You may use either Scala (spark-shell) or Python (pyspark). These instructions assume you are using Python.
2. Review the simple text file we will be using by viewing (without editing) the file in a text editor. The file is frostroad.txt.
3. Define an RDD to be created by reading in a simple text file:

```
pyspark> mydata = sc.textFile(“file://frostroad.txt")
```

Load and View Text File

4. Note that Spark has not yet read the file. It will not do so until you perform an operation on the RDD. Try counting the number of lines in the dataset:

```
pyspark> mydata.count()
```

   The count operation causes the RDD to be materialized (created and populated). The number of lines should be displayed, e.g.
   Out[4]: 23

5. Try executing the collect operation to display the data in the RDD. Note that this returns and displays the entire dataset. This is convenient for very small RDDs like this one, but be careful using collect for more typical large datasets.

```
pyspark> mydata.collect()
```

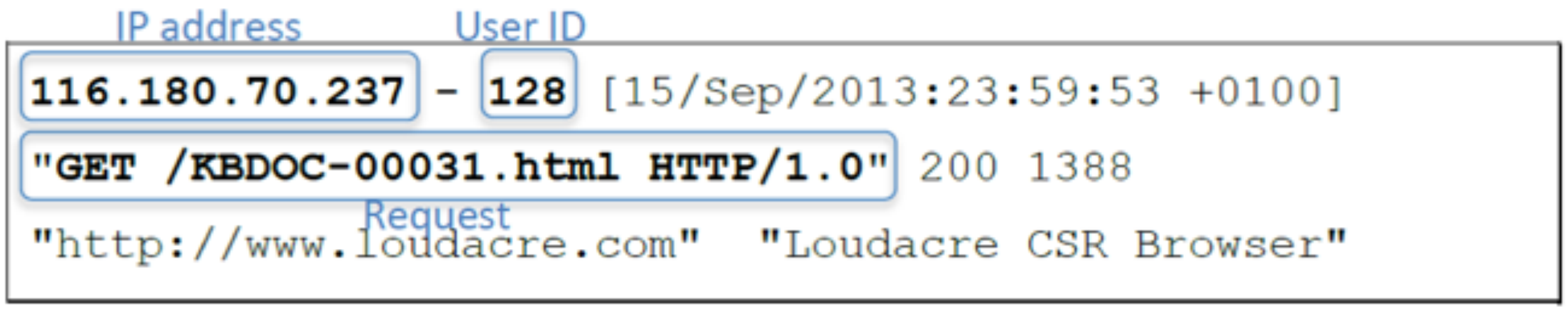
Load and View Text File

6. Using command completion, you can see all the available transformations and operations you can perform on an RDD. Type mydata. and then the [TAB] key.

   Explore the web log files

   In this exercise you will be using data in ~/training_materials/sparkdev/data/weblogs. Initially you will work with the log file from a single day. Later you will work with the full data set consisting of many days worth of logs.

7. Review one of the .log files in the directory. Note the format of the lines, e.g.



```
116.180.70.237 - 128 [15/Sep/2013:23:59:53 +0100]
"GET /KBDOC-00031.html HTTP/1.0" 200 1388
"http://www.loudacre.com"  "Loudacre CSR Browser"
```

Load and View Textfile

8. Set a variable for the data file so you do not have to retype it each time. Note the home directory is incorrect!

```
pyspark> logfile="file://sparkdev/data/weblogs/2013-09-15.log"
```

9. Create an RDD from the data file.

```
pyspark> logs = sc.textFile(logfile)
```

10. Create an RDD containing only those lines that are requests for JPG files.

```
pyspark> jpglogs=logs.filter(lambda x: ".jpg" in x)
```

11. View the first 10 lines of the data using take:

```
pyspark> jpglogs.take(10)
```

Load and View Textfile

12. Sometimes you do not need to store intermediate data in a variable, in which case you can combine the steps into a single line of code. For instance, if all you need is to count the number of JPG requests, you can execute this in a single command:

```
pyspark> sc.textFile(logfile).filter(lambda x: ".jpg" in x).count()
```

13. Now try using the map function to define a new RDD. Start with a very simple map that returns the length of each line in the log file.

```
pyspark> logs.map(lambda s: len(s)).take(5)
```

    This prints out an array of five integers corresponding to the length of each of the first five lines in the log file.

14. That’s not very useful. Instead, try mapping to an array of words for each line:

```
pyspark> logs.map(lambda s: s.split()).take(5)
```

    This time it prints out five arrays, each containing the words in the corresponding log file line.

Load and View Textfile

15. Now that you know how map works, define a new RDD containing just the IP addresses from each line in the log file. (The IP address is the first "word" in each line).

```
pyspark> ips = logs.map(lambda s: s.split()[0])
pyspark> ips.take(5)
```

16. Although take and collect are useful ways to look at data in an RDD, their output is not very readable. Fortunately, though, they return arrays, which you can iterate through:

```
pyspark> for x in ips.take(10): print x
```

17. Finally, save the list of IP addresses as a text file:

```
pyspark> ips.saveAsTextFile("file://iplist")
```

18. In a terminal window, list the contents of the /home/training/iplist folder. You should see multiple files. The one you care about is part-0000, which should contain the list of IP addresses. "Part" (partition) files are numbered because there may be results from multiple tasks running on the cluster; you will learn more about this later.

If You Have More Time

If you have more time, attempt the following challenges:

1. Challenge 1: As you did in the previous step, save a list of IP addresses, but this time, use the whole web log data set (weblogs/*) instead of a single day s log.
   - Tip: You can use the upNarrow to edit and execute previous commands. You should only need to modify the lines that read and save the files. Note that the directory you specify when calling saveAsTextFile() must not already exist.
2. Challenge 2: Use RDD transformations to create a dataset consisting of the IP address and corresponding user ID for each request for an HTML file. (Disregard requests for other file types). The user ID is the third field in each log file line. Display the data in the form *ipaddress/userid*, e.g.:

```
165.32.101.206/8
100.219.90.44/102
182.4.148.56/173
246.241.6.175/45395
```

Review the API Documentation for RDD Operations

Visit the Spark API page you bookmarked previously. Follow the link at the top for the RDD class and review the list of available methods.

Working with Pair RDDs

Pair RDDS

This time, work with the entire set of data files in the weblog folder rather than just a single day's logs.

1. Using MapReduce, count the number of requests from each user.
   a) Use map to create a Pair RDD with the user ID as the key, and the integer 1 as the value. (The user ID is the third field in each line.) Your data will look something like this:

(userid,1)
(userid,1)
(userid,1)
…

Pair RDDS

b) Use reduce to sum the values for each user ID. Your RDD data will be similar to:

(userid,5)
(userid,7)
(userid,2)
…

2. **Display the user IDs and hit count for the users with the 10 highest hit counts.**

a) Use map to reverse the key and value like this:

(5,userid)
(7,userid)
(2,userid)
…

b) Use sortByKey (False) to sort the swapped data by count

Pair RDDS

5. Create an RDD where the user id is the key, and the value is the list of all the IP addresses that user has connected from. (IP address is the first field in each request line.)
   – Hint: Map to (userid, ipaddress) and then use groupByKey.

(*userid*,20.1.34.55)
(*userid*,245.33.1.1)
(*userid*,*65.50.196.141*)
…

(*userid*,[20.1.34.55, 74.125.239.98])
(*userid*,[75.175.32.10, 245.33.1.1,
(*userid*,[65.50.196.141])
…

Pair RDDS

4. The data set in the /sparkdev/data/accounts.csv consists of information about Loudacre's user accounts. The first field in each line is the User ID, which corresponds to the user ID in the web server logs. The other fields include account details such as creation date, first and last name and so on.
   Join the accounts data with the weblog data to produce a dataset keyed by user ID which contains the user account information and the number of website hits for that user.
   a) Map the accounts data to key/valueNlist pairs: (userid, [values…])

(*userid1*,[*userid1*,2008-11-24 10:04:08,\N,Cheryl,West,4905 Olive Street,San
(*userid2*,[*userid2*,2008-11-23 14:05:07,\N,Elizabeth,Kerns,4703 Eva Pearl
(*userid3*,[*userid3*,2008-11-02 17:12:12,2013-07-18 16:42:36,Melissa,Roman,3539 James Martin
…

Pair RDDs

b) Join the Pair RDD with the set of userid/hit counts calculated in the first step. (Note: the example data below is abbreviated, and you should see the actual userids display)

```
(userid1,([2012-06-17
05:14:00, 2013-11-28 19:26:31, Elaine, Robinson, 3019
Romano Street, Stockton, CA, …], 42))
(userid2,([2013-09-07
21:43:37, 2014-01-09 17:45:57, Ricky, Pope, 4535 Highland
Drive, Sacramento, CA,…],2))
(userid3,([2013-07-21
05:02:38, 2014-03-04 01:10:23, Elizabeth, Macdonald, 632
Marcus Street, Oakland, CA,…],30))
…
```

Pair RDDs

c) Display the user ID, hit count , and first name (3rd value) and last name (4th value) for the first 10 elements, e.g.:

```
userid1 4 Cheryl West
userid2 8 Elizabeth Kerns
userid3 1 Melissa Roman
```

If You Have More Time

If you have more time, attempt the following challenges:

1. Challenge 1: Use keyBy to create an RD 1. D of account data with the postal code (9th field in the CSV file) as the key.
   - Hint: refer to the Spark API for more information on the keyBy operation
   - Tip: Assign this new RDD to a variable for use in the next challenge

2. Challenge 2: Create a pair RDD with postal code as the key and a list of names (Last Name,First Name) in that postal code as the value.
   - Hint: First name and last name are the 4th and 5th fields respectively
   - Optional: Try using the mapValues operation

If You Have More Time

3. Challenge 3: Sort the data by postal code, then for the first five postal codes, display the code and list the names in that postal zone, e.g.

```
Jenkins,Thad
Rick,Edward
Lindsay,Ivy
…
--- 85004
Morris,Eric
Reiser,Hazel
Gregg,Alicia
Preston,Elizabeth
```

Using HDFS

Exploring HDFS

1. Open a terminal window (if one is not already open) by double-clicking the Terminal icon on the desktop.
2. Most of your interaction with the system will be through a command-line wrapper called hdfs. If you run this program with no arguments, it prints a help message. To try this, run the following command in a terminal window:

```
$ hdfs
```

3. The hdfs command is subdivided into several subsystems. The subsystem for working with the files on the cluster is called FsShell. This subsystem can be invoked with the command hdfs dfs. In the terminal window, enter:

```
$ hdfs dfs
```

- You see a help message describing all the commands associated with the shell subsystem.

Exploring HDFS

```
$ hdfs dfs -ls /
```

4. **Enter**

```
$ hdfs dfs -ls /user
```

   This shows you the contents of the root directory in HDFS. There will be multiple entries, one of which is /user. Individual users have a “home” directory under this directory, named after their username.

5. Try viewing the contents of the /user directory by running:

```
$ hdfs dfs -ls /user/*
```

   You will see your home directory in the directory listing.

6. List the contents of your home directory by running:

Exploring HDFS

- There are no files yet, so the command silently exits. This is different than if you ran hdfs dfs -ls /foo, which refers to a directory that doesn’t exist and which would display an error message.
- Note that the directory structure in HDFS has nothing to do with the directory structure of the local filesystem; they are completely separate namespaces.

Uploading Files

Besides browsing the existing filesystem, another important thing you can do with FsShell is to upload new data into HDFS.

Change directories to the local filesystem directory containing the sample data for the course.

```
$ cd /sparkdev/data
```

If you perform a regular Linux ls command in this directory, you will see a few files, including the weblogs directory you used in previous exercises.

1. Insert this directory into HDFS:

```
$ hdfs dfs -put weblogs /weblogs
```

   This copies the local weblogs directory and its contents into a remote HDFS directory named /weblogs.

Uploading Files

9. List the contents of your HDFS home directory now:

```
$ hdfs dfs -ls /
```

   You should see an entry for the weblogs directory.

Viewing and manipulating Files

Now view some of the data you just copied into HDFS

11. **Enter**

```
$ hdfs dfs -cat weblogs/2014-03-08.log | tail -n 50
```

    This prints the last 50 lines of the file to your terminal. This command is useful for viewing the output of Spark programs. Often, an individual output file is very large, making it inconvenient to view the entire file in the terminal. For this reason, it is often a good idea to pipe the output of the fs -cat command into head, tail, more, or less.

12. To download a file to work with on the local filesystem 12. use the dfs -get command. This command takes two arguments: an HDFS path and a local path. It copies the HDFS contents into the local filesystem:

```
$ hdfs dfs -get weblogs/2013-09-22.log ~/logfile.txt
$ less ~/logfile.txt
```

Viewing and manipulating Files

- There are several other operations available with the hdfs dfs command to perform most common filesystem manipulations: mv, rm, cp, mkdir, and so on. Enter:

```
$ hdfs dfs
```

  This displays a brief usage report of the commands available within FsShell. Try playing around with a few of these commands.

Accessing HDFS files in Spark

14. Start the spark shell as the HDFS user. In the Spark Shell, create an RDD based on one of the files you uploaded to HDFS. For example:

```
pyspark> logs=sc.textFile("hdfs://localhost/weblogs/2014-03-08.log")
```

15. Save the JPG requests in the dataset to HDFS:

```
pyspark> logs.filter(lambda s: ".jpg" in s).\
saveAsTextFile("hdfs://localhost/user/adminuser/jpgs")
```

16. Back in the terminal, view the created directory and files it contains

```
$ hdfs dfs -ls jpgs
$ hdfs dfs -cat jpgs/* | more
```

17. *Optional*: Explore the NameNode UI: http://localhost:50070 . In particular, try menu selection **Utilities Browse'the'file'system**.

Running Spark Shell on a Cluster

Start the Spark Standalone Cluster

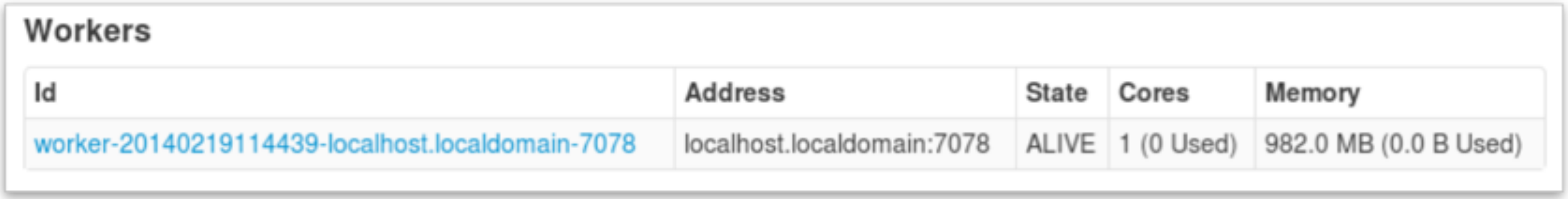
1. In a terminal window, start the Spark Master and Spark Worker daemons:
   As root…

```
<spark install>/spark/sbin/start-all.sh
```

Note: You can stop the services by replacing start with stop

View the Spark Standalone Cluster UI

2. Start a browser and visit the Spark Master UI at http://localhost:8080/.
3. You should not see any applications in the Running Applications or Completed Applications areas because you have not run any applications on the cluster yet.
4. A real world Spark cluster would have several workers configured. In this class we have just one, running locally, which is named by the date it started, the host it is running on, and the port it is listening on. For example:



Workers

Id	Address	State	Cores	Memory
worker-20140219114439-localhost.localdomain-7078	localhost.localdomain:7078	ALIVE	1 (0 Used)	982.0 MB (0.0 B Used)

View the Spark Standalone Cluster UI (1)

5. Click on the worker ID link to view the Spark Worker UI and note that there are no executors currently running on the node.
6. Return to the Spark Master UI and take note of the URL shown at the top. You may wish to select and copy it into your clipboard.

Start Spark Shell on the cluster

7. Return to your terminal window and exit Spark Shell if it is still running.
8. Start Spark Shell again, this time setting the MASTER environment variable to the Master URL you noted in the Spark Standalone Web UI.

   Or the Scala shell:
   spark-shell —master spark://<host name>:7077

   You will see additional info messages confirming registration with the Spark Master. (You may need to hit Enter a few times to clear the screen log and see the shell prompt.)

Start Spark Shell on the cluster

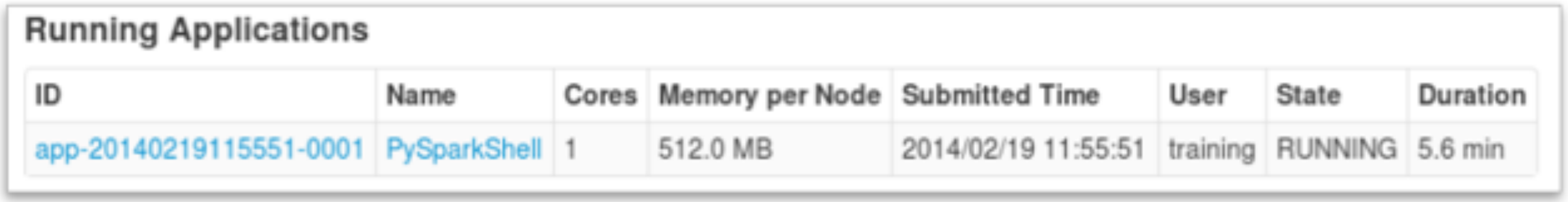
9. You can confirm that you are connected to the correct master by viewing the sc.master property:

```
pyspark> sc.master
```

10. Execute a simple operation to test execution on the cluster. For example,

```
pyspark> sc.textFile(“hdfs:///weblogs/*”).count()
```

11. Reload the Spark Standalone Master UI and note that now the Spark Shell appears in the list of running applications.



Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20140219115551-0001	PySparkShell	1	512.0 MB	2014/02/19 11:55:51	training	RUNNING	5.6 min

12. Click on the application ID (app-*xxxxxxx*) to see an overview of the application, including the list of executors running (or waiting to run) tasks from this application. In our small classroom cluster, there is just one, running on the single node in the cluster, but