MIDS W205

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab #** | 4 | **Lab Title** | An Introduction to Apache Spark and Spark SQL. |
| **Related Module(s)** | 6 | **Goal** | Get you started on Spark, Pyspark, and Spark SQL |
| **Last Updated** | 9/29/15 | **Expected duration** | 60-90 minutes |

# Introduction

Apache Spark is an open source distributed computing framework. Spark uses in-memory processing and can in some situations be up to 100 times faster than regular Hadoop MapReduce. The core of the Spark architecture is the concept of RDDs. You should read about them and why they enable scale and resiliency in a distributed environment with unreliable nodes.

Apache Spark has a basic computing substrate and several frameworks built on top of it. There is a framework for querying structured data (Spark SQL), an analytics framework for microbatching (they call it streaming), a machine-learning framework, and so on. In this lab you will be learning about basic RDDs as well as about Spark SQL.



Spark SQL assumes that data are structured according to a relational model, which enables us to use SQL to query the data. It provides a programming abstraction based on what is called DataFrames. Spark SQL acts as a distributed SQL query engine.

Spark can be used with several different programming languages. We will be using Python as our preferred way of interacting with Spark. Spark is commonly utilized in a programmatic way. There are also command-line interfaces (CLIs) that provide convenient ways of interactively using RDDs. In this lab we will use both.

# Instructions, Resources, and Prerequisites

You can perform this lab on one of the course-provided AMIs, but you need to have Hadoop set up properly. Follow the instructions provided in other sections of the course. You can also perform it on another computer, such as your laptop, by installing Spark from the link in the table below.

Since the AMIs use CDH 5.4.5 and Spark SQL is not officially supported on that version, you may encounter problems with the CLI portion of the lab. If you have problems, you can skip the Spark SQL CLI portion (Step 5 and Step 6), but you still need to try the programmatic way of using Spark SQL shown in Step 7.

|  |  |
| --- | --- |
| **Resource** | **What** |
| Spark download (for your own installation) | http://spark.apache.org/downloads.html |
| http://spark.apache.org/docs/ latest/programming-guide.html | This guide shows each of the features in each of Spark’s supported languages. It is easiest to follow if you launch Spark’s interactive shell.  This guide includes references to the basic commands you can perform on RDDs, such as filter records, count records, and join of datasets. |
| https://spark.apache.org/docs/1.1.0/sql-programming-guide.html | Guide for using Spark SQL programmatically. |
| https://spark.apache.org/docs/ latest/sql-programming-guide.html#running-the-spark-sql-cli | Guide to Spark SQL CLI Shell. |
| https://spark.apache.org/docs/0.9.0/python-programming-guide.html | Python Spark programming guide. For operations on RDD’s see the Scala programming guide. |
| https://spark.apache.org/docs/0.9.0/scala-programming-guide.html | This is the scala programming guide. Look here for transformations and actions on Spark RDD’s. |
| http://spark.apache.org/docs/ latest/api/scala/index.html#org.apache.spark.SparkContext | Programming guide for the Spark Context object. Here you can find actions available on the Spark Contexts. |

# Step 0. Check Installation and Prepare Data

In this section we will help you understand the basics of your Spark installation. We will also help you download and assemble a dataset we will use later in the lab. Downloading the dataset takes a couple of steps because it is reasonably large and stored in GitHub. GitHub has limitations on file size, so we needed to split the dataset into manageable chunks to upload it. Your instructor may provide you with an alternative way of getting the same dataset.

First, if an example starts with the “$” prompt, it is run in the Linux shell. If it starts with the “>>>” prompt, it is run in pyspark shell.

Echo your SPARK\_HOME environment variable to see where your Spark is installed.

$ echo $SPARK\_HOME

You should see something like the below. Observe that this may differ depending on were you are running this lab.

$ echo $SPARK\_HOME

/user/lib/spark

Navigate to that directory, and examine the contents.

$ cd $SPARK\_HOME

$ ls

CHANGES.txt NOTICE README.md bin data ec2 lib python

LICENSE R RELEASE conf derby.log examples old sbin

If you look in bin you will see spark-shell, pyspark, and other tools for running and managing Spark.

In your shell profile (often .bash\_profile), you may see the following lines that indicate that the Spark commands are in your shell execution path.

export SPARK=/usr/lib/spark

export SPARK\_HOME=$SPARK

export PATH=$SPARK/bin:$PATH

You can confirm that your shell can find them by running the Unix/Linux that command.

$ which spark-shell

$ which pyspark

$ which spark-sql

If your shell cannot find any of those programs, you will likely need to check your installation. If it can find the commands, it will return the location of the programs.

We will be using two different datasets in this lab. One is a weblog dataset; the other is a dataset with historic crime data from 2001 for the Chicago area. If you have time to play around with the dataset, it is interesting to explore types of crimes, density of crimes, and so forth. For this lab you can use any substantial data file as a replacement for “Crimes\_-\_2001\_to\_present.csv”, but you need to modify the commands accordingly. You can download the “Crimes\_-\_2001\_to\_present.csv” file from GitHub. Because it is a large file, we had to compress and split the file when we uploaded it to GitHub. Therefore, you will need to merge the parts and uncompress the result.

The instructors may have the files more easily accessible for you if you have problems with the procedure below.

Retrieve the files from this directory on GitHub: <https://github.com/UC-Berkeley-I-School/w205-labs-exercises/tree/master/data/Crimes_-_2001_to_present_data>

*Note: the URL may differ for you as we create a new repository each semester.*

You can retrieve them by cloning the exercise repository, if you have already done so, us a git pull to update. A clone would look like the following.

git clone https://github.com/UC-Berkeley-I-School/w205-labs-exercises.git

The files are in the data/Crimes\_-\_2001\_to\_present\_data directory.

Once you have the files (there should be seven of them, all starting with the letter *x*), run the following commands (make sure you do not have other files starting with *x* in the directory). The first one will concatenate the split files into one file. The original file was a compressed CSV file, so we name it appropriately. Next we uncompress it to get the original CSV file.

$ cat x\* > Crimes\_-\_2001\_to\_present.csv.gz

$ gunzip Crimes\_-\_2001\_to\_present.csv.gz

The result should be that you have the Crime data CSV file in your directory. You also still have the original split files. If you run ls it should look something like this:

$ ls

Crimes\_-\_2001\_to\_present.csv xac xaf

xaa xad xag

xab xae

You can check the correctness of the resulting files by checking the file size or the number of rows in the file. The size may vary on different computers, but the number of lines should remain the same.

$ du -s Crimes\_-\_2001\_to\_present.csv

2688712 Crimes\_-\_2001\_to\_present.csv

$ wc -l Crimes\_-\_2001\_to\_present.csv

5862796 Crimes\_-\_2001\_to\_present.csv

You could remove the split files using rm, but it is better to use the –i option so that you do not accidentally remove other files in the directory that you want to save.

$ rm -i x\*

At this point you should be able to see the Crime data file in your directory.

$ ls

Crimes\_-\_2001\_to\_present.csv

Now you are good to go. In addition, you can also try some useful Linux/Unix commands such as du, wc, and cat.

## What You Should Have Learned

You should have learned the basic installation of Spark. You have also verified that the Spark programs can be found by your interactive Linux (Unix) shell.

# Step 1. Start pyspark

First we will start a Spark shell so that we can access Spark and interactively process Spark commands. We will be using pyspark, which is a Python-based shell for Spark.

Assuming you have the Spark bin directory in your PATH environment variable, you can start pyspark by issuing the following command:

$pyspark

Otherwise go to the /bin directory in the installation ($SPARK\_HOME) folder and type the following:

$./pyspark

In the pyspark shell you can use Python instructions and create RDDs. You can also apply operations on RDDs. Create a Python variable with some value using the following command (this is plain old Python):

>>> x = [1,2,3,4,5,6,7,8,9];

>>> print x

[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> len(x)

9

A Spark context is the “object” you use to refer to the Spark cluster. Type sc to verify you have one already in your Python shell:

>>> sc

<pyspark.context.SparkContext object at 0x1063b3410>

So far you have used only Python statements and checked that you have a Spark context. You can use the Spark context to create a Spark RDD from Python data using the command parallelize.

>>> distData = sc.parallelize(x);

>>> print distData;

ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:391

The Spark context parallelize action is used to take local programming collections and create RDDs from them. In this case, we created a RDD from a Python array.

The resulting value of the distData variable is an RDD representation. Try running len(distData); what happens, and why? To count elements in an RDD you need to use RDD actions. You can find a list of available actions in the programming guide. To count the elements, you use the count() action. Try the following:

>>> nx=distData.count()

>>> print nx

9

As you probably noticed, the default level of logging can be distracting. To reduce the volume of logging information, go to the $SPARK\_HOME/conf directory. Sometimes this directory is stored in another location. If you are using one of our AMIs, look in the /usr/lib/spark/conf/ directory. Create a log4j.properties file. If you do not already have one, you can get one by copying the log4j.properties.template file. To set the logging level to warnings (WARN), change INFO to WARN in the property log4j.rootCategory=WARN console.

# Set everything to be logged to the console

log4j.rootCategory=WARN, console

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.target=System.err

log4j.appender.console.layout=org.apache.log4j.PatternLayout

Restart pyspark, and rerun the commands above. You should see much less output in the Spark shell.

## What You Should Have Learned

You should understand how to start pyspark. You also learned that pyspark is a shell with access to the Python language and to the underlying Spark cluster capabilities. We also showed you how to make pyspark less verbose by reducing the amount of logging displayed. If you have problems, you may consider increasing logging again to understand what is going on in your execution.

# Step 2. Load a File and Count the Rows

Spark is commonly used to process large sets of data, and naturally we often read these data files from disk. The action textFile is an operation on Spark contexts that creates a RDD from a file that resides in HDFS or the local file system. Make sure you cloned the data file Crimes\_-\_2001\_to\_present.csv in the lab directory (assuming you are using the directory “/data/mylab”). Now create an RDD from that file using the following action:

crimedata =sc.textFile("file:///data/mylab/Crimes\_-\_2001\_to\_present.csv")

If you run the AIM, you can copy the data to HDFS and have pyspark get the file from there.

Print the number of lines in this RDD using the following command:

>>> print crimedata.count()

5862796

>>>

## As you can see there are almost 6 million records, so it took a few seconds to count them.

You can get the first element of the RDD with the first operation:

>>> crimedata.first()

You can get the *n* first elements with the operation take(n):

>>> crimedata.take(10)

As you see the data include the header information. Removing the header from the RDD is not straightforward because the RDD is immutable by design. One way to remove the header is to create a new RDD and to filter out the first row. An example of one of the many ways to do this follows:

>>> noHeaderCrimedata = crimedata.zipWithIndex().filter(lambda (row,index): index > 0).keys()

It is a quite a processing-heavy way of doing it, but as we mentioned, RDDs are immutable. You cannot just go in and remove a record. This is an inherent characteristic of Spark, which allows it to do certain things more efficiently.

Print the first line to verify that the header is gone, and count the lines to ensure the number seems correct.

>>> noHeaderCrimedata.first()

>>> noHeaderCrimedata.count()

Here is a more Python-like way of doing the same task that may be easier to understand. We first define a Python function in our Python Spark shell using the following line:

>>> def remove\_header(itr\_index, itr): return iter(list(itr)[1:]) if itr\_index == 0 else itr

We then execute a mapPartitionWithIndex operation, passing that function as an argument.

>>> noHeaderCrimeData2 = crimedata.mapPartitionsWithIndex(remove\_header)

Print the first line and the count to make sure they are correct.

## What You Should Have Learned

You should understand how to create an RDD from a file and how apply operations on the RDD. You used examples such as first and count. We also illustrated that RDDs are immutable and that even to remove one row (the header) you need to create a new RDD.

# Step 3. Filter Records and Structures

One obvious operation for Spark is to filter the data. Let’s filter out all crimes that seem to be related to NARCOTICS.

We can do this using the filter operation and a lambda function that checks for the word “NARCOTICS” in each row. We will return only rows that included that word.

narcoticsCrimes = noHeaderCrimedata.filter(lambda x: "NARCOTICS" in x)

>>> narcoticsCrimes.count()

663712

It appears that 663,712 crimes are related to narcotics. Use take(n) to check that the data seems good. For example:

>>> narcoticsCrimes.take(20)

The RDD we have is just long strings. The fact that fields are comma separated does not mean anything to the Spark RDD. If we want to perform more advanced computations and manipulations, we need to parse the rows and create the appropriate structure for our data. The following operation splits each record as an array using the Python operation split. It then creates a new RDD where each row is an array of strings as opposed to one long string.

>>> narcoticsCrimeRecords = narcoticsCrimes.map(lambda r : r.split(","))

You can see the first array record using

>>> narcoticsCrimeRecords.first()

You can verify that you still have the same number of rows using

>>> narcoticsCrimeRecords.count()

## What You Should Have Learned

You should understand that RDDs are immutable. You can filter RDDs, but doing so creates a new RDD. You should also understand that RDDs do not understand anything about the structure of the records (except for key value structures, which we discuss in the next section). But you can store any Python structure that seems useful in an RDD.

# Step 4. Key Values

An important structure in Spark is called a key value pair. In Python these are represented as Python tuples. A tuple is an immutable sequence of elements of various types.

You can create a new RDD consisting of tuples using the following operation:

>>> narcoticsCrimeTuples = narcoticsCrimes.map(lambda x: (x.split(",")[0], x))

You can check that the number of tuples is the same as the number of records in the data.

>>> narcoticsCrimeTuples.count()

This operation takes the first element and makes it a key, and the rest of the row becomes the value part of the tuple. You can examine the tuple using the following operation:

>>> narcoticsCrimeTuples.first()

And you can check it out using these RDD and Python functions.

Get the first tuple:

>>> firstTuple=narcoticsCrimeTuples.first()

How many elements do you have in the tuple?

>>> len(firstTuple)

What is the key of the first tuple?

>>> firstTuple[0]

What is the value of the first tuple?

>>> firstTuple[1]

There is one little problem with the tuple. Can you spot it? How should we change the map and lambda functions above to address that?

You can perform many operations once you have a key value tuple. You can join, reduce, map, and so on. You can read about the operations in the *RDD Spark Programming Guide*. One operation you can do is to sort by key:

>>> sorted=narcoticsCrimeTuples.sortByKey()

If you print the first element in the sorted RDD and the original RDD, you will see they are different.

>>> sorted.first()

>>> narcoticsCrimeTuples.first()

## What You Should Have Learned

Now you should understand the concept of key value tuples and how you can create them. You have also tried one operation on RDDs using the key value structure.

# Step 5. Start Spark SQL

***Important****: You can skip this step if you run on an AMI that does not have appropriate support for Spark SQL. If you have a later version of Spark with Hadoop installed on your computer, you should be able to run this. You can also try the steps to install a complete Spark/Hadoop per instructions on the troubleshooting section. Proceed to Step 7 to learn about Spark SQL API.*

Spark SQL can be used directly from pyspark or a scala shell. But there is also a Spark SQL CLI called the Beeline client. If you use pyspark or use Spark SQL programmatically, you need create a special Spark SQL context. With the Spark SQL CLI, the context is already there for you and you can use SQL commands.

You start Beeline with the following command:

$spark-sql

Once started you can run some commands to ensure that it works. Show tables, create a table, and drop the table by running the next commands:

spark-sql> show tables;

spark-sql> create table dummy (somedata varchar(500));

OK

Time taken: 0.369 seconds

spark-sql> show tables;

dummy false

Time taken: 0.053 seconds, Fetched 1 row(s)

spark-sql> drop table dummy;

spark-sql> show tables;

## What You Should Have Learned

You should understand the difference between using the Spark SQL CLI and using Spark SQL programmatically. You are able to start Spark SQL CLI and issue some basic commands to see that it works.

# Step 6. Spark SQL Table Loaded With Data From a CSV file

***Important****: You can skip this step if you run on an AMI that does not have appropriate support for Spark SQL. If you have a later version of Spark with Hadoop installed on your computer, you should be able to run this. You can also try the steps to install a complete Spark/Hadoop per instructions on the troubleshooting section. Proceed to Step 7 to learn about Spark SQL API.*

You can create a Spark SQL table. The following create statement creates a table that has a schema that corresponds to the web\_session\_log data. Run the following create statement directly on the spark-sql shell prompt:

create table Web\_Session\_Log

(DATETIME varchar(500),

USERID varchar(500),

SESSIONID varchar(500),

PRODUCTID varchar(500),

REFERERURL varchar(500))

row format delimited fields terminated by ‘\t’

stored as textfile;

You can load files from the local files system or from HDFS. Let’s load weblog data available on GitHub and should have been cloned with the data directory.

Run the describe command to see that it was created correctly. If it was not, you may need to drop it and try again correcting any mistakes.

spark-sql> describe web\_session\_log;

datetime varchar(500) NULL

userid varchar(500) NULL

sessionid varchar(500) NULL

productid varchar(500) NULL

refererurl varchar(500) NULL

Time taken: 0.083 seconds, Fetched 5 row(s)

Assuming you have the weblog data in the directory where you are running the spark-sql shell, you can load the file from the file system into the table using the command below. If the file is located somewhere else, you need to modify the path to the file.

spark-sql> LOAD DATA LOCAL INPATH "./weblog\_lab.csv" INTO TABLE web\_session\_log;

Once the data are loaded you can count the number of rows using the following select statement:

spark-sql> select count(\*) from web\_session\_log;

You can check that this count seems reasonable by comparing it with the number of rows in the original file. You can get the number of files using the Unix/Linux command wc.

$ wc -l weblog\_lab.csv

Using Spark SQL you can use a number of SQL commands to query your data. Let’s select all rows in the weblog that are related to eBay.

spark-sql> select \* from web\_session\_log where refererurl = "http://www.ebay.com" ;

And now let’s count the number of rows that are related to eBay.

spark-sql> select count(\*) from web\_session\_log where refererurl = "http://www.ebay.com" ;

## What You Should Have Learned

In this section you should have learned how to create a table in the Spark SQL CLI and how to load data into the empty table. You also practiced some simple SQL commands on the loaded dataset.

# Step 7. Accessing Spark SQL in Python Code

Spark SQL can be used from a program or directly from a shell such as pyspark. Using a shell requires that you create the appropriate Spark context and programmatically define schemas and such. A table is ultimately represented as a Spark DataFrame. Next we will go through step by step what you need to import, how you read the data, how you create an object that represents the schema, and, finally, how you combine the schema definition and the data to create a table DataFrame.

We also show some simple queries using SQL in the resulting SQL DataFrame.

First make sure to import all necessary file types to your Python environment. You can try the following example interactively in a pyspark shell. Make sure you run it in the directory where you store the data file or adapt the path in the example accordingly.

$ pyspark

>>> from pyspark.sql import SQLContext

>>> from pyspark.sql.types import \*

Create the Spark SQL Context.

>>> sqlContext = SQLContext(sc)

Read the weblog data into an RDD. You may need to adjust the path to the data based on where you stored them.

>>> lines = sc.textFile('file:///data/labs/w205-labs-exercises/data/weblog\_lab.csv ')

Create a map of the data so that they can be structured into a table.

>>> parts = lines.map(lambda l: l.split('\t'))

>>> Web\_Session\_Log = parts.map(lambda p: (p[0], p[1],p[2], p[3],p[4]))

Create string with the name of the columns of your table.

>>> schemaString = 'DATETIME USERID SESSIONID PRODUCTID REFERERURL'

Create a data structure of StructFields that can be used to create a table.

>>> fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

Combine the fields into a schema object.

>>> schema = StructType(fields)

Create a table based on a DataFrame using the data that was read and the structure representing the schema.

>>> schemaWebData = sqlContext.createDataFrame(Web\_Session\_Log, schema)

Register the object as a table with a table name.

>>> schemaWebData.registerTempTable('Web\_Session\_Log')

Query the table.

>>> results = sqlContext.sql('SELECT count(\*) FROM Web\_Session\_Log')

Use the DataFrame operation show to print the content of the result of the query.

>>> results.show()

The following query can be used to query the number of rows related to eBay:

select count(\*) from web\_session\_log where REFERERURL'= "http://www.ebay.com" ;

Note: The case sensitivity is sometime different, depending on where you run the command. Keep that in mind if you see certain errors.

Enhance the script to execute the above query, and answer Submission 2.

SUBMISSION 2: Submit the number of rows returned by the select on eBay entries.

It should print that the result has one cell with some value.

SUBMISSION 3: Submit the number returned in the DataFrame of the result.show() command above.

Another query, with a screen shot.

>>> results = sqlContext.sql('SELECT \* FROM Web\_Session\_Log')

>>> results.show()

See below for screenshot of what you should see.



You can also run the program as a script. Let’s assume you create a script mysql.py, which you placed in /tmp with the data file. The content of mysql.py follows. You may need to adjust the location of data file based on where you stored it and if you opted to have it on HDFS.

from pyspark import SparkContext

from pyspark.sql import SQLContext

from pyspark.sql.types import \*

sc = SparkContext("local", "weblog app")

sqlContext = SQLContext(sc)

lines = sc.textFile('file:///data/labs/w205-labs-exercises/data//weblog\_lab.csv')

parts = lines.map(lambda l: l.split('\t'))

Web\_Session\_Log = parts.map(lambda p: (p[0], p[1],p[2], p[3],p[4]))

schemaString = 'DATETIME USERID SESSIONID PRODUCTID REFERERURL'

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

schema = StructType(fields)

schemaWebData = sqlContext.createDataFrame(Web\_Session\_Log, schema)

schemaWebData.registerTempTable('web\_session\_log')

results = sqlContext.sql('SELECT \* FROM web\_session\_log')

results.show()

You can now run the Python Spark SQL script using the command. You can use pyspark to run the command, but the recommended way is to use spark-submit.

$ spark-submit /tmp/mysql.py

The output may look something like the following screenshot.



## What You Should Have Learned

You should have learned how to create a Python script that uses Spark SQL and how to run the script.

# Step 8. Caching Tables and Uncaching Tables

## Caching Tables

This process is extremely useful when you are joining a tiny dataset with a huge dataset.

CACHE TABLE and UNCACHE TABLE statements are available to do this task, making it very easy.

To cache a table:

CACHE TABLE logs\_last\_month;

To uncache a table:

UNCACHE TABLE logs\_last\_month;

Once a table is cached, you can use in your Spark queries.

# Submissions, summary

There are three items that need to be submitted from this lab to be approved.

In Step 3:

SUBMISSION 1: Submit the first 10 rows of the unsorted and the sorted RDD to show that you successfully created both. Also explain the issue with the tuple and how to possibly correct it in the map/lambda function that was used to create the tuples.

In Step 5:

SUBMISSION 2: Submit the number of rows returned by the select on eBay entries.

In Step 6:

SUBMISSION 3: Submit the number returned in the DataFrame of the result.show() command.

# Troubleshooting

**Connection problem on laptop or Macbook**

If you get an exception similar to “ERROR SparkContext: Error initializing SparkContext. java.net.UnknownHostException:…”, ensure you have the brand and model of your computer added to the /etc/hosts file. For example add the line “127.0.0.1 <myhost>”, where <myhost> is the name of your computer.

7

**Hadoop connection problem on EC2 instance**

If you get an error that looks something like this on the AMI:

py4j.protocol.Py4JJavaError: An error occurred while calling o34.collect.

: java.net.ConnectException: Call From ip-10-61-206-219.ec2.internal/10.61.206.219 to localhost:8020 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: <http://wiki.apache.org/hadoop/ConnectionRefused>

your Hadoop instance is likely not running. With the “ucb\_w205\_complete - ami-71cdb014,” use the start-hadoop.sh and stop-hadoop.sh scripts to start and stop the Hadoop service.

**AMI**

If you are using the AMI, you need to attach an EBS volume.

Follow these instructions for how to create a volume and attach it to you instance:

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-attaching-volume.html>

When you create a volume you give it a name. I used /dev/sdh (h for hadoop) so that I remember it. When you check volumes on you EC2 instance, you will see different name. In the example below my sdh got named xvdh. I recognized it by the h and the size.

[root@ip-10-61-206-219 ~]# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda1 202:1 0 10G 0 disk /

xvdb 202:16 0 4G 0 disk

xvdh 202:112 0 100G 0 disk /data

You have an attached volume. Now you need to create a file system on that raw disk volume and then mount the volume on your root file system so that it can be accessed by programs. A file system is like a tree of trees, and mounting essentially means you attach a tree to a branch of another tree. File systems are also a structure of information on your disk that tracks files, blocks of files, who owns the files, and so forth. That is why you need to create the file system.

This guide explains how to create and mount the file system. The only problem I noticed with the guide was that it left out that you should provide the type of file system to mount. If the example in the guide fails for you, try this:

sudo mount -t ext4 /dev/xvdh /data

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-using-volumes.html>

## Error at ‘\t’

Sometimes cutting and pasting changes the representation of characters. Edit the command in spark-sql and make sure you have ‘’ quotes.

## Local Spark Installation for spark-sql CLI

You create a new installation of Spark using the below instructions you will be able to run the CLI. The following if for the AMI, but you can do the corresponding steps on you own computer or in another environment.

wget http://www.us.apache.org/dist/spark/spark-1.5.0/spark-1.5.0-bin-hadoop2.6.tgzmv spark-1.5.0-bin-hadoop2.6 spark15

export SPARK\_HOME=$HOME/spark15

export HADOOP\_CONF\_DIR=/etc/hadoop/conf

I would also propose changing Log tracing to WARN in the conf/log4j file.

If you check which spark-sql you use you should see the following.

# which spark-sql

/root/spark15/bin/spark-sql

You need to make sure Hadoop is started. Then start spark-sql and you should be able to do step-5 and step-6 of this lab.

## In sufficient memory

If you get an ERROR concerning insufficient memory you should probably make sure you get more memory for the node you are running the lab on. This is particularly important so that you can process somewhat realistic data sets for Exercises and Projects. To get through this lab, however, you can reduce the data set. Lets assume you had issues with the Crime data set. This data set has close to 6 million rows. Lets say you can handle 1 million rows in memory on your node. You could then use the following command to split the data set in 1 million row parts. The split command will create a number of splits. The xaa files is the first part of the original file. We want to use that as it contains the header information that we look at later in the lab.

$ split -l 1000000 Crimes\_-\_2001\_to\_present.csv

$ ls

Crimes\_-\_2001\_to\_present.csv xad

xaa xae

xab xaf

xac

$ mv xaa Crimes\_-\_2001\_to\_present\_1M-1.csv

You can now complete the pyspark sections on the lab using this data set. Be aware that some of the counts etc. mentioned in the lab may be incorrect with respect to this smaller data set.