

# Programming with HDFS and Spark

## Accessing HDFS using Python

### Prerequisite:

Please have already install python as part of your spark installation. You may even install python in your local host e.g. Windows machine as well separately for running Pycharm.

### Installing python

As we are using Python 3.5 for our local machine and Python 3. 6 is not supported for Spark 2.0.0, hence better to think to use Python 3.5 as well for our Spark & Linux Virtual Machine.

Hope you download Python-3.5.2.tgz e.g. in /downloads. Now let us install the python manually.

```
# cd /downloads  
# ls -l Python-3.5.2.tgz  
# sudo tar xzf Python-3.5.2.tgz  
# cd Python-3.5.2
```

Let us install a compiler now and couple of recommend packages e.g. SSL/TSL support for Pip.

```
# sudo yum install gcc  
# sudo yum install zlib-devel bzip2-devel sqlite sqlite-devel openssl-devel  
# sudo ./configure --prefix=/usr/local/  
# sudo make altinstall  
# /usr/local/bin/python3.5 --version
```

Note: We are doing alternate installation so that original python version used for system is intact.

You can use pip3.5 to install hdfs package for your desired python version.

```
#pip3.5 --version
```

The next steps are to change the Python version in two places i.e. Spark Environmental Shell as well as hadoop user's shell properties.

Configure python 3.5 for Spark and hadoop User Environmental variable

## Big Data & Hadoop Hands On Training Material

```
# vim /usr/local/spark/conf/spark-env.sh
```

Update following line:

```
export PYTHONPATH="/usr/local/lib/python3.5/site-packages/:$PYTHONPATH"
```

```
export PYSARK_PYTHON=/usr/local/bin/python3.5
```

Configure python 3.5 for hadoop user profile.

```
#vim /home/hadoop/.bashrc
```

```
# Set Environment For Python 3 in required user .bashrc
```

```
export PATH="/usr/local/spark/bin:/usr/local/bin:/usr/local/lib/python3.5/site-packages:$PATH"
```

```
#Python for Spark
```

```
export PYTHONPATH="/usr/local/lib/python3.5/site-packages:$PYTHONPATH"
```

```
export PYSARK_PYTHON=/usr/local/bin/python3.5
```

```
# User specific aliases and functions
```

```
alias python=/usr/local/bin/python3.5
```

```
alias python3=/usr/local/bin/python3.5
```

Update your user profile

```
# source ~/.bashrc
```

Now you are good to go with python 3.5 with Spark 2.0.0. If your spark is running during your configuration, Please restart the process to get the updated spark environment configuration.

```
# /usr/local/spark/sbin/stop-all.sh
```

```
# /usr/local/spark/sbin/start-all.sh
```

```
#pyspark
```

### Installing hdfs python package

Please use pip3.5 to install hdfs package for your desired python version.

```
# sudo pip3 install hdfs
```

Create a file named .hdfscli.cfg in your user home directory and configure HDFS connection details as follows:

## Big Data & Hadoop Hands On Training Material

```
# cd /home/hadoop
```

```
# vim .hdfscli.cfg
```

```
[global]
```

```
default.alias = bdrenhdfs
```

```
[bdrenhdfs.alias]
```

```
uri = http://bdrenfdludcf01:50070
```

```
user = hadoop
```

Now you should be able to import hdfs to perform file operations for your HDFS cluster.

Important Note: In case of your windows user, please do same for in user home directory. Better to use IP address of your namenode incase of external host unless it is configured in DNS or in your local PC hosts file.

### Accessing and Perform HDFS programmatically

Consider you logged in your linux virtual machine and you would like to perform actions from there to your HDFS.

```
# vim accesshdfslinux.py
```

```
client = Config().get_client('bdrenhdfs')
```

```
files = client.list('/mydir')
```

```
count = len(files)
```

```
print(count)
```

```
print("####Files are in HDFS###")
```

```
for filename in files:
```

```
    print(filename)
```

```
#You can do many operations using client object e.g. rename, creating directory etc.
```

```
conn=InsecureClient('http://192.168.0.106:50070', user='hadoop')
```

```
conn.download('/mydir/trips.csv','/tmp/',overwrite=True)
```

```
conn.upload("/", "/home/hadoop/development/trips.csv",overwrite=True)
```

## Big Data & Hadoop Hands On Training Material

```
#Work with OS Command Utility

os.system("/usr/local/hadoop/bin/hadoop fs -put
/home/hadoop/development/corrected /")

#Work with command output

cmdoutput = subprocess.check_output("/usr/local/hadoop/bin/hadoop fs -ls /mydir",
shell=True)

print("Sub process command output" + str(cmdoutput))

#Traversing local file system

main_dir_name="/home/hadoop/development"

print("####Files are in Local File Systems###")

for dir_name, subdirs, files in os.walk(main_dir_name):

    for filename in files:

        if filename.endswith(".csv"):

            print(filename)

            #cmd= "/home/hadoop/Hadoop/bin/hadoop fs -put " + filename + " /" #It will
work when you install hdfs binaries in your local machine

            #os.system(cmd) #Similarly you can run any command by using the technique
```

Consider you logged in your local machine e.g. Windows and you would like to perform actions from there to your HDFS.

### **accesshdfs.py**

```
from hdfs import Config
from hdfs import InsecureClient
import os
import subprocess

if __name__ == '__main__':

    client = Config().get_client('bdrenhdfs')
    files = client.list('/mydir')
    count = len(files)
    print(count)
```

## Big Data & Hadoop Hands On Training Material

```
print("####Files are in HDFS###")

for filename in files:
    print(filename)

#You can do many operations using client object e.g. rename, creating directory etc.
#conn=InsecureClient('http://192.168.0.106:50070', user='hadoop')
#conn.download('/mydir/trips.csv', 'D:\')
#conn.upload("/", "D:\\Track\\Datasciencelab\\BDREN\\Software_Binaries\\trips.csv")

#Work with OS Command Utility
#os.system("ping bdrenfdludcf01")
#Work with command output
cmdoutput = subprocess.check_output("ipconfig")
print("Sub process command output" + str(cmdoutput))

#exit()

#Traversing local file system
main_dir_name="D:\\Track\\Datasciencelab\\BDREN\\Software_Binaries"
print("####Files are in Local File Systems###")
for dir_name, subdirs, files in os.walk(main_dir_name):
    for filename in files:
        if filename.endswith(".csv"):
            print(filename)
            #cmd= "/home/hadoop/Hadoop/bin/hadoop fs -put " + filename + "/" #It will
work when you install hdfs binaries in your local machine
            #os.system(cmd) #Similarly you can run any command by using the technique
```

## Submitting Spark Jobs

### Prerequisite:

You have installed your spark cluster and can able to run pyspark utility interactively.

Login as hadoop user and start running pyspark

```
#pyspark
```

## Big Data & Hadoop Hands On Training Material

```
>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>>
df_covid=sqlContext.read.format("csv").option("header",'true').load("hdfs://bdrenfdludc
f01:9000/covid-19")
>>> df_covid.show(10)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|
dateRep|day|month|year|cases|deaths|countriesAndTerritories|geold|countryterritor
yCode|popData2018|
+-----+-----+-----+-----+-----+-----+-----+-----+
|18/04/2020|18|4|2020|51|1|Afghanistan|AF|AFG|
37172386|
|17/04/2020|17|4|2020|10|4|Afghanistan|AF|AFG|
37172386|
|16/04/2020|16|4|2020|70|2|Afghanistan|AF|AFG|
37172386|
|15/04/2020|15|4|2020|49|2|Afghanistan|AF|AFG|
37172386|
|14/04/2020|14|4|2020|58|3|Afghanistan|AF|AFG|
37172386|
|13/04/2020|13|4|2020|52|0|Afghanistan|AF|AFG|
37172386|
|12/04/2020|12|4|2020|34|3|Afghanistan|AF|AFG|
37172386|
|11/04/2020|11|4|2020|37|0|Afghanistan|AF|AFG|
37172386|
|10/04/2020|10|4|2020|61|1|Afghanistan|AF|AFG|
37172386|
|09/04/2020|9|4|2020|56|3|Afghanistan|AF|AFG|
37172386|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 10 rows
```

## Big Data & Hadoop Hands On Training Material

```
>>> df_covid
```

```
DataFrame[dateRep: string, day: string, month: string, year: string, cases: string, deaths: string, countriesAndTerritories: string, geold: string, countryterritoryCode: string, popData2018: string]
```

```
>>> df_covid.createOrReplaceTempView("covid")
```

```
>>> sqlDF = sqlContext.sql("SELECT sum(cases) TotalCases, sum(deaths) TotalDeaths, countriesAndTerritories FROM covid group by countriesAndTerritories")
```

```
>>> sqlDF.show(2)
```

```
+-----+-----+-----+
|TotalCases|TotalDeaths|countriesAndTerritories|
+-----+-----+-----+
| 33.0| 0.0| Chad|
| 3.0| 0.0| Anguilla|
+-----+-----+-----+
```

```
only showing top 2 rows
```

```
>>> sqlDF.filter(sqlDF["countriesAndTerritories"]=="Bangladesh").show()
```

```
>>> sqlDF.coalesce(1).write.mode("overwrite").csv("/home/hadoop/development/c.csv")
```

So now we can run the above transformation and action from an interactive shell. In real scenario, we would need to run all types of transformations and actions from a backend script which will run may be on-demand or in schedule. Let us make a spark python script which can be submitted to cluster manager to run in different supported modes.

```
[hadoop@bdrenfdludcf01 development]$ vim dataprocessingonspark.py
```

```
from pyspark import SparkConf, SparkContext, SQLContext
from pyspark.sql import SQLContext
from random import randint
from time import sleep
from pyspark.sql.session import SparkSession
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
ch = logging.StreamHandler()
logger.addHandler(ch)
```

## Big Data & Hadoop Hands On Training Material

```
import sys
def dataprocessing(filePath, sqlContext):
    logger.info( "Entering into Function")
    df_covid = sqlContext.read.format("csv").option("header", 'true').load(filePath)
    logger.info( '#####DataSet has:%s' , df_covid.count())
    sleepInterval = randint(10,100)
    logger.info( '#####Sleeping for %s' , sleepInterval)
    sleep(sleepInterval)
    df_covid.createOrReplaceTempView("covid")
    sqlDF = sqlContext.sql(
        "SELECT sum(cases) TotalCases, sum(deaths) TotalDeaths, countriesAndTerritories
FROM covid group by countriesAndTerritories")

sqlDF.coalesce(1).write.mode("overwrite").csv("/home/hadoop/development/covidanalysis.csv")
    #sqlDF.filter(sqlDF["countriesAndTerritories"] == "Bangladesh").show()

if __name__ == '__main__':
    # You can run it as spark-submit dataprocessingonspark.py myfilename.csv local (or spark://....)

    #if len(sys.argv) < 2:
    # print ('Usage dataprocessingonspark.py <filename> <sparkMasterHost>')
    # sys.exit(0)

    print(sys.argv)
    filename = "covid-19"
    #filename = sys.argv[1]
    #iterations = int(sys.argv[2])
    #sparkHost = sys.argv[2]
    logger.info('-----')
    logger.info('Filename:%s', filename)
    #logger.info('Iterations:%s', iterations )
    #logger.info('SparkMaster:%s', sparkHost)
    #logger.info('-----')

    basePath = 'hdfs://bdrenfdludcf01:9000/'
    absFilePath = basePath + filename

    logger.info( '.....Starting spark.....Loading from %s ' , filename)
    logger.info( 'Starting up....')
    # Configure Spark
```



## Big Data & Hadoop Hands On Training Material

```
#conf = SparkConf().setAppName(APP_NAME).set("spark.default.parallelism",  
"8").set("spark.cores.max", "5").set("spark.executor.cores",  
"2").set("spark.driver.memory", "2g").set("spark.executor.memory", "4g").set(  
"spark.storage.memoryFraction", "0.4").set("spark.local.dir", "/tmp/4GNOKIA/")  
#conf = conf.setMaster("spark://172.26.7.192:7077 ") # We need to set master as  
spark URL in command-line to run in multiple machine  
#sc = SparkContext(conf=conf)  
#sqlContext = SQLContext(sc)  
  
#sc = SparkSession.builder.config("spark.cores.max",  
"5").config("spark.executor.cores", "2").config("spark.driver.memory",  
"8g").config("spark.executor.memory", "4g").appName("Data  
Processing").getOrCreate()  
sc = SparkSession.builder.appName("Spark Data Processing").getOrCreate()  
logger.info ('Initializing sqlContext')  
sqlContext = SQLContext(sc)  
dataprocessing(absFilePath, sqlContext)
```

### Submitting Applications

The spark-submit script in Spark's bin directory is used to launch applications on a cluster. It can use all of Spark's supported [cluster managers](#) through a uniform interface so you don't have to configure your application especially for each one.

### Running Spark Job in Local mode

```
# cd /home/user/development/  
  
# ls -l dataprocessingonspark.py  
  
# /usr/local/spark/bin/spark-submit --master local[1] dataprocessingonspark.py
```

Note: local[1] means it will request one CPU.

### Running Spark Job in Standalone mode

```
# /usr/local/spark/bin/spark-submit --master spark://bdrenfdludcf01:7077  
dataprocessingonspark.py
```

### Running Spark Job in Yarn

Please check whether your Yarn process is running using following command:

```
#jps
```

## Big Data & Hadoop Hands On Training Material

```
$ /usr/local/spark/bin/spark-submit --master yarn --conf  
spark.yarn.submit.waitAppCompletion=false dataprocessingonspark.py
```

Spark can support Apache MESOS and Kubernetes Cluster to work as a Cluster Manager for Spark Job. For more details of spark job submission, please look into following URL:

<https://spark.apache.org/docs/latest/submitting-applications.html>

### Supporting Applications

Monitoring Spark Process: <http://192.168.0.106:4040>

Monitoring Spark Management Interface: <http://192.168.0.106:9999>

Yarn Management Interface: <http://192.168.0.106:8088/cluster>

### More Data frame Operation Examples

The following operations have been discussed:

Data frame's Schema:

```
>>>df.printSchema()
```

OR

```
>>>df
```

Column Name Changing:

```
>>>from pyspark.sql.functions import col
```

```
>>>df_CM.select(col("BTS_ID").alias("BTS_ID_CM"),  
col("SEG_ID").alias("SEG_ID_CM")).show()
```

Dropping a Column:

```
>>>df_CM.drop("BTS_ID").show()
```

How to add a new Column with a pre-defined value:

## Big Data & Hadoop Hands On Training Material

```
from pyspark import SparkConf, SparkContext, SQLContext
```

```
from pyspark.sql.functions import lit
```

```
>>>df_CM_raw_update = df_CM_raw_tmp.withColumn('Measurement_Period',  
lit('20161113080000'))
```

```
>>>join_complete = join.withColumn('Brand_Name', lit('Uber'))
```

How to keep only distinct rows in a dataframe:

```
>>>df_epsbear_raw = df_epsbear_raw.distinct()
```

How to repartition or split a dataframe and RDD

```
>>>df_traffic_tmp.repartition(5)
```

```
>>> numbers = sc.parallelize([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],7) #Doing 7 partitions.
```

Note: You can avoid java.outOfMemory (OOM) error by splitting a Data frame.

Loading multiple similar Data frame:

```
>>>df_epsbear_raw = df_epsbear_raw.unionAll(df_epsbear_tmp)
```

For more please look Spark related class reference.