Homework 6: Spark Streaming

E-63 Big Data Analytics
Harvard University, Autumn 2017

Tim Hagmann

Oktober 14, 2017

Problem 1 (20%)

Question: Lecture notes contain script *network-count.py* in both Spark Streaming API and Spark Structured Streaming API. Use Linux *nc (NetCat)* utility to demonstrate that scripts work. Run both scripts on your own VM with Spark 2.2 installation. Cloudera VM with Spark 1.6 does not have Spark Structured Streaming API.

Note: I'm running problem 1, 4 and 5 on the same EC2 instance as in HW3 (see HW3 for the setup instructions). Problem 2 and 3 are run on the cloudera quickstart docker images. The setup instructions can be found inside problem 2.

i) Spark Streaming API

Before beeing able to work with the *network-count.py* we have to start spark.

```
sudo spark/sbin/start-all.sh
```

```
starting org.apache.spark.deploy.master.Master, logging to
/home/tim/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1-ip-172-31-24-35.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to
/home/tim/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-ip-172-31-24-35.out
```

Next we're running the *network-count.py* in one window and the *nc -lk 9999* in another window. Inside the *NetCat* Server we can type interactivily while the *network-count.py* performs a count.

```
# Netcat server
nc -lk 9999

# Network count
spark-submit --master local[4] \
/home/tim/e63-coursework/hw6/scripts/network-count.py localhost 9999
```

Netcat server

In the *NetCat* server we can input again the following text:

```
tim@ip-172-31-24-35:~/e63-coursework/hw6/data$ nc -lk 9999
hello world

this is a test
talking to the world
```

Network count

The network count produces with the above input the following output:

```
-------------------------------------------
Time: 2017-10-13 14:46:45
-------------------------------------------
(u'world', 1)
(u'hello', 1)

-------------------------------------------
Time: 2017-10-13 14:46:48
-------------------------------------------

-------------------------------------------
Time: 2017-10-13 14:46:51
-------------------------------------------
(u'a', 2)
(u'this', 1)
(u'test', 1)
(u'to', 1)
(u'world', 2)
(u'talking', 1)
(u'the', 2)
(u'is', 1)
...

-------
```

As can be seen above, the *network-count.py* script appears to be working. It correctly counts the *hello world* line as well as the other two lines.

ii) Spark Structured Streaming API

In order to run on the *Spark Structured Streaming API* we have to use the *network-count-structured.py*. The *NetCat* server is still running, that means we don't have to start it again.

```
spark-submit --master local[4] \
/home/tim/e63-coursework/hw6/scripts/network-count-structured.py localhost 9999
```

Netcat server

In the *NetCat* server we can input again the same text as seen above.

```
tim@ip-172-31-24-35:~/e63-coursework/hw6/data$ nc -lk 9999
hello world

this is a test
talking to the world
```

Network count structured

The network count produces with the above input the following output:

```
tim@ip-172-31-24-35:~/e63-coursework$ spark-submit --master local[4]
/home/tim/e63-coursework/hw6/scripts/network-count-structured.py localhost 9999
-------------------------------------------
Batch: 0
-------------------------------------------

[Stage 1:====>                                                    (15 + 4) / 200]
[Stage 1:======>                                                  (22 + 5) / 200]
[Stage 1:========>                                                (30 + 4) / 200]
[Stage 1:============>                                            (40 + 4) / 200]
[Stage 1:==============>                                          (50 + 4) / 200]
[Stage 1:=================>                                       (62 + 4) / 200]
[Stage 1:====================>                                    (74 + 4) / 200]
[Stage 1:========================>                                (84 + 4) / 200]
[Stage 1:============================>                            (97 + 4) / 200]
[Stage 1:==============================>                         (112 + 4) / 200]
[Stage 1:==================================>                     (123 + 4) / 200]
[Stage 1:======================================>                 (137 + 4) / 200]
[Stage 1:==========================================>             (151 + 4) / 200]
[Stage 1:==============================================>         (164 + 4) / 200]
[Stage 1:==================================================>     (178 + 4) / 200]
[Stage 1:======================================================>  (191 + 4) / 200]

+-----+-----+
| word|count|
+-----+-----+
|hello|    1|
|world|    1|
+-----+-----+

-------------------------------------------
Batch: 1
-------------------------------------------

[Stage 5:============>                                            (41 + 4) / 200]
[Stage 5:================>                                        (54 + 4) / 200]
[Stage 5:====================>                                    (70 + 4) / 200]
[Stage 5:========================>                                (86 + 4) / 200]
[Stage 5:============================>                           (102 + 4) / 200]
[Stage 5:================================>                       (117 + 4) / 200]
[Stage 5:====================================>                   (133 + 4) / 200]
[Stage 5:========================================>               (145 + 4) / 200]
[Stage 5:============================================>           (161 + 4) / 200]
[Stage 5:================================================>       (176 + 4) / 200]
[Stage 5:====================================================>   (189 + 4) / 200]

+-----+-----+
| word|count|
+-----+-----+
|hello|    1|
|   is|    1|
|world|    1|
|    a|    1|
| this|    1|
```

```
| test|    1|
+-----+-----+

-------------------------------------------
Batch: 2
-------------------------------------------

[Stage 9:============>                                            (46 + 4) / 200]
[Stage 9:=================>                                       (62 + 4) / 200]
[Stage 9:======================>                                  (79 + 4) / 200]
[Stage 9:============================>                            (94 + 4) / 200]
[Stage 9:=============================>                          (107 + 4) / 200]
[Stage 9:==================================>                     (120 + 4) / 200]
[Stage 9:=======================================>                (138 + 4) / 200]
[Stage 9:============================================>           (153 + 4) / 200]
[Stage 9:=================================================>      (170 + 4) / 200]
[Stage 9:=====================================================>  (186 + 4) / 200]

+-------+-----+
|   word|count|
+-------+-----+
|  hello|    1|
|talking|    1|
|     is|    1|
|    the|    1|
|  world|    2|
|      a|    1|
|   this|    1|
|   test|    1|
|     to|    1|
+-------+-----+
```

The *network-count-structured.py* is also working. It is running in batches and splits the analysis.

Problem 2 (30%)

Question: Run the following experiments on Cloudera VM. You need HDFS for these programs to run.

i) WSL Setup (optional)

A lot of company notebook don't allow running virtual machines (*VT-X* is disabled). This is also the case with my notebook. An alternative of using VM is to use *docker*. After installing docker nativily, it is possible to run docker against the *WSL* engine (Windows Subsystem for Linux). This is because Docker can expose a TCP endpoint which the CLI (i.e., WSL) can attach to. This step is optional. On a *Mac* or *Linux* machine docker can be nativily called from the shell.

Note: The TCP endpoint on Windows is turned off by default. To activate it, right-click the Docker icon in your taskbar and choose Settings, and tick the box next to "Expose daemon on tcp://localhost:2375 without TLS".

Add docker to WSL (optional)

With that done, all we need to do is instruct the CLI under Bash to connect to the engine running under Windows instead of to the non-existing engine running under Bash, like this:

```
echo "export DOCKER_HOST='tcp://0.0.0.0:2375'" >> ~/.bashrc
source ~/.bashrc
```

ii) Docker setup

The docker image can be pulled directly from the shell.

```
# Install CDH 5.12
docker pull cloudera/quickstart:latest

# Start docker
docker run --hostname=quickstart.cloudera --privileged=true -it --rm -i -t -p \
8888:8888 -p 80:80 -p 7180:7180 -p 7077:7077 -p 7078:7078 -p 8080:8080 -p \
18080:18080 -p 18081:18081 -p 8020:8020 cloudera/quickstart \
/usr/bin/docker-quickstart
```

Start second docker

It is possible to start a multitute of sessions. In order to perform the following analysis a second terminal window has to be started. However, in order to login we first need container image id.

```
# Get id name
docker ps
```

```
CONTAINER ID        IMAGE                  COMMAND                  CREATED
9cacc524be9f        cloudera/quickstart    "/usr/bin/docker-q..."   2 minutes ago

STATUS              PORTS
Up 2 minutes        0.0.0.0:80->80/tcp, 0.0.0.0:7180->7180/tcp, 0.0.0.0:8080->8080/tcp, ...
```

With the above information we can login to docker.

```
# Login with a second screen
docker exec -it --user root 9cacc524be9f bash
```

Change to cloudera user

The docker image starts by default into the root user. We're changing to the installed cloudera user (password=cloudera).

```
su cloudera
cd ~
```

iii) Setup Image

Before beeing able to start with the analysis we have to setup the docker environment variables.

```
# Add environment variables
echo "export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera" >> /home/cloudera/.bash_profile
echo "export SPARK_HOME=/usr/lib/spark" >> /home/cloudera/.bash_profile
echo "export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce" >> /home/cloudera/.bash_profile
echo "export PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:$SPARK_HOME/bin" >>
/home/cloudera/.bash_profile

# Source file
source /home/cloudera/.bash_profile
```

Next we're installing *pip* and *findspark*.

```
sudo yum install python-pip
sudo pip install --upgrade pip
sudo pip install findspark
```

Modify log4j.properties

In order to eliminate the error messages in the *spark-shell*, we have to create the *log4j.properties* file from the template file and setting the log level to *ERROR* and *log4j.rootCategory=ERROR*.

```
# Create log4j.properties
sudo mv /usr/lib/spark/conf/log4j.properties.template /usr/lib/spark/conf/log4j.properties

# Edit file
sudo vi /usr/lib/spark/conf/log4j.properties

# Replace INFO, WARN with ERROR.
:g /WARN/s//ERROR/g
:g /INFO/s//ERROR/g
:wq
```

Start spark

Next we have to start spark on the machine. The option *localhost[2]* indicates that 2 threads are beeing used.

```
sudo /usr/lib/spark/sbin/start-all.sh --host localhost[2]
```

Check status

For the following step we're checking if all the necessary *hadoop* services are running.

```
for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x status ; done
```

```
[root@quickstart /]# for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x
status ; done
Hadoop datanode is running                                 [  OK  ]
Hadoop journalnode is running                              [  OK  ]
Hadoop namenode is running                                 [  OK  ]
Hadoop secondarynamenode is running                        [  OK  ]
Hadoop httpfs is running                                   [  OK  ]
Hadoop historyserver is running                            [  OK  ]
Hadoop nodemanager is running                              [  OK  ]
```

```
Hadoop proxyserver is not running                          [FAILED]
Hadoop resourcemanager is running                          [  OK  ]
```

The above output shows, that all the necessary hadoop services are up and running. The proxy service is not necessary to run for the following exercices.

iv) Get data

Question: Expand provide *orders.tar.gz* file. Also, download shell scrips *splitAndSend.original.sh* and *splitAndSend.sh* and the Python script *count-buys.py*.

```
git clone https://github.com/greenore/e63.git
cd e63

# Untar
tar xvzf orders.tar.gz
```

Setup hadoop directory structure

In order for the above scripts to work, we have to create the necessary directories inside the *HDFS*.

```
# Create 3 directories in hdfs
hadoop fs -mkdir -p staging input output

hadoop fs -ls
```

```
[cloudera@quickstart e63]$ hadoop fs -ls
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2017-10-13 07:41 stating
drwxr-xr-x   - cloudera cloudera          0 2017-10-13 07:41 input
drwxr-xr-x   - cloudera cloudera          0 2017-10-13 07:41 output
```

v) SplitAndSend Original

Question: First run *splitAndSend.original.sh* and *count-buys.py*. Record the failure mode of *count-buys.py*. Simply read the error message produced and tell us what is happening.

```
# Run scripts
chmod +x splitAndSend.original.sh
./splitAndSend.original.sh input
```

```
spark-submit --master local[2] /home/cloudera/e63/count-buys.py
```

```
[cloudera@quickstart ~]$ spark-submit --master local[2] /home/cloudera/e63/count-buys.py
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Killed
```

The SLF4J error can be ignored. However, the process is beeing killed. In order to understand why, let us look at the content of the different folders.

Show content

```
hadoop fs -ls output
```

```
[cloudera@quickstart e63]$ hadoop fs -ls output
Found 16 items
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930326000.txt
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930329000.txt
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930332000.txt
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930335000.txt
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930338000.txt
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:06 output/output-1507930341000.txt
...
```

The output folder look good. There are different files inside the *HDFS*. Next we're looking at the staging folder.

```
hadoop fs -ls staging
```

```
[cloudera@quickstart ~]$ hadoop fs -ls staging
```

As expected, there is nothing in the staging folder. The next step is to look at the input folder.

```
hadoop fs -ls input
```

```
Found 12 items
-rw-r--r--   1 cloudera cloudera     437626 2017-10-13 21:07 input/chunkaa
-rw-r--r--   1 cloudera cloudera     448647 2017-10-13 21:07 input/chunkab
-rw-r--r--   1 cloudera cloudera     448605 2017-10-13 21:07 input/chunkac
-rw-r--r--   1 cloudera cloudera     448624 2017-10-13 21:07 input/chunkae
-rw-r--r--   1 cloudera cloudera     448553 2017-10-13 21:07 input/chunkaf
-rw-r--r--   1 cloudera cloudera     448436 2017-10-13 21:08 input/chunkag
-rw-r--r--   1 cloudera cloudera     448679 2017-10-13 21:08 input/chunkah
-rw-r--r--   1 cloudera cloudera     448424 2017-10-13 21:08 input/chunkai
-rw-r--r--   1 cloudera cloudera     448564 2017-10-13 21:08 input/chunkaj
-rw-r--r--   1 cloudera cloudera     458595 2017-10-13 21:08 input/chunkak
-rw-r--r--   1 cloudera cloudera     458580 2017-10-13 21:08 input/chunkal
-rw-r--r--   1 cloudera cloudera          0 2017-10-13 21:08 input/chunkam._COPYING_
```

There is a chunkam._COPYING file inside the input folder. That means that by copying directly from the local file system to the hadoop *input* directory, there are cases where Spark tries to read the file just after it's name was changes. This leads to files named chunkam._COPYING. When the file is finished copying, the extension is changed. If Spark now sees the new temp file and then tries to read it just after its name was changed, it causes a *file not found* error and the scripts exits. That means that once one of the two tasks finishes successfully, the other task is killed. This is a case of the so called *race condition*.

vi) Eliminate race condition

Question: Then run script *splitAndSend.sh* and Python program *count-buys.py* and tell us what the results are. In both cases show use contents of your HDFS directories input, output and staging. The second script *splitAndSend.sh* is supposed to reduce or eliminate the race condition.

The problem of the race condition can be fixed by staging files instead of moving theme directly into the input directory. The command *hadoop fs -mv staging/$f input* creates an atomic file that solves the race condition. The problem can also be decreased when inserting a 1 second delay.

Empty folders

First we're empty the HDFS folders.

```
# Empty folders
hadoop fs -rm -r staging/*
hadoop fs -rm -r output/*
hadoop fs -rm -r input/*
```

Run scripts

Next we can run the *splitAndSend.sh* script with the adopted changes described above.

```
# Run scripts
chmod +x splitAndSend.sh
./splitAndSend.sh input
```

```
spark-submit --master local[2] /home/cloudera/e63/count-buys.py
```

Show content

```
hadoop fs -ls staging
```

```
[cloudera@quickstart ~]$ hadoop fs -ls staging
Found 1 items
-rw-r--r--   1 cloudera cloudera     448564 2017-10-13 21:13 staging/chunkaj
```

The staging folder now holds 1 file before moving.

```
hadoop fs -ls input
```

```
[cloudera@quickstart ~]$ hadoop fs -ls input
Found 12 items
-rw-r--r--   1 cloudera cloudera     437626 2017-10-13 21:12 input/chunkaa
-rw-r--r--   1 cloudera cloudera     448647 2017-10-13 21:12 input/chunkab
-rw-r--r--   1 cloudera cloudera     448605 2017-10-13 21:12 input/chunkac
-rw-r--r--   1 cloudera cloudera     448794 2017-10-13 21:12 input/chunkad
-rw-r--r--   1 cloudera cloudera     448624 2017-10-13 21:12 input/chunkae
-rw-r--r--   1 cloudera cloudera     448553 2017-10-13 21:12 input/chunkaf
-rw-r--r--   1 cloudera cloudera     448436 2017-10-13 21:12 input/chunkag
-rw-r--r--   1 cloudera cloudera     448679 2017-10-13 21:13 input/chunkah
-rw-r--r--   1 cloudera cloudera     448424 2017-10-13 21:13 input/chunkai
-rw-r--r--   1 cloudera cloudera     448564 2017-10-13 21:13 input/chunkaj
-rw-r--r--   1 cloudera cloudera     458595 2017-10-13 21:13 input/chunkak
-rw-r--r--   1 cloudera cloudera     458580 2017-10-13 21:14 input/chunkal
```

The input folder looks good.

```
hadoop fs -ls -R output
```

```
[cloudera@quickstart ~]$ hadoop fs -ls -R output
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:12 output/output-1507929150000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929150000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929150000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:12 output/output-1507929153000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929153000.txt/_SUCCESS
```

```
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929153000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:12 output/output-1507929156000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929156000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929156000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:12 output/output-1507929159000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:12 output/output-1507929159000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera   29 2017-10-13 21:12 output/output-1507929159000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:12
[...]
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:14 output/output-1507929249000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929249000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929249000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:14 output/output-1507929252000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929252000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929252000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:14 output/output-1507929255000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929255000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929255000.txt/part-00000
drwxr-xr-x   - cloudera cloudera    0 2017-10-13 21:14 output/output-1507929258000.txt
-rw-r--r--   1 cloudera cloudera    0 2017-10-13 21:14 output/output-1507929258000.txt/_SUCCESS
-rw-r--r--   1 cloudera cloudera   29 2017-10-13 21:14 output/output-1507929258000.txt/part-00000
```

As can be seen from above, the scripts run without issues. All the results are correctly written to the output folder.

vii) Show part-00000

Question: In the second run, locate an output file named *part-00000* that is not empty and show its content to us.

The following command gives as the output of a non-empty *part-00000* file.

```
hadoop fs -cat output/output-1507929258000.txt/part-00000
```

```
[cloudera@quickstart e63]$ hadoop fs -cat output/output-1507929258000.txt/part-00000
(False, 4876L)
(True, 5124L)
```

The above output means, that there were 5124 buys and 4876 sells in that particualar 3 second interval.

Problem 3 (10%)

Question: In the second run of the previous problem you will notice that many of *part-00000* files in your output directory are empty. Could you explain why.

The *part-00000* files are micro-batches. Those not-empty contain the result of counting in the particular 3 second interval (see description above). That means, empty files result when there was no input in that particular interval.

Problem 4 (20%)

Question: Could you rewrite count-buys.sh in Spark Structured Streaming API. If you do that change script splitAndSend.sh to move generated chunks from the local files system directory staging to local file system

directory input. Run this experiment on your VM with Spark 2.2.

i) Rewrite splitAndSend

In order to use the native *Spark Structured Streaming API* we need to rewrite *splitAndSend.sh* so that it runs locally and no longer on the *HDFS* system. In order to do that first the two folder *input* and *staging* have to be created on the local filesystem.

```
### Generate folders
mkdir -p /home/tim/e63-coursework/hw6/data/input/
mkdir -p /home/tim/e63-coursework/hw6/data/staging
```

Next we can rewrite *splitAndSend.sh* to *splitAndSend2.sh*

```
#!/bin/bash

# Split data into chunks
split -l 10000 /home/tim/e63-coursework/hw6/data/orders.txt \
/home/tim/e63-coursework/hw6/data/staging/chunk

# Simulate stream with 3 seconds difference
for f in `ls /home/tim/e63-coursework/hw6/data/staging/chunk*`; do
        sleep 3
        mv $f /home/tim/e63-coursework/hw6/data/input/
        rm -f $f
done
```

ii) Rewrite count-buys

Next we have to rewrite the count-buys function. In principal what we have to do is to define the schema for the input rows, create the spark session, create a dataframe representing the stream of input files and count the buys / sells in the window using the timestamp as a watermark. The resulting file can be found in *count-buys2.py*. It's code is below:

```
# Libraries
from __future__ import print_function
import findspark
findspark.init("/home/tim/spark")
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Build session
spark = SparkSession.builder.appName("StructuredCountBuys").getOrCreate()

# Define schema
schema = StructType([StructField('time', TimestampType(), True),
                     StructField('orderId', IntegerType(), True),
                     StructField('clientId', IntegerType(), True),
                     StructField('symbol', StringType(), True),
                     StructField('amount', IntegerType(), True),
                     StructField('price', FloatType(), True),
```

```
                    StructField('buy', StringType(), True)])

# Read stream
df_stockmarket = spark.readStream.csv("/home/tim/e63-coursework/hw6/data/input/",
                                      schema=schema, sep=',')

# Group by (with watermark and window)
df_buys = df_stockmarket.withWatermark("time", "1 minutes") \
                        .groupBy("buy", window("time", "1 seconds")) \
                        .count()

# Write stream
df_buys.writeStream.queryName("aggregates") \
       .outputMode("complete") \
       .format("console") \
       .start() \
       .awaitTermination()
```

iii) Run Spark API Stream

Next we can run the code.

SplitandSend

```
# Empty folders (optional)
# rm -r /home/tim/e63-coursework/hw6/data/staging/*
# rm -r /home/tim/e63-coursework/hw6/data/input/*

# Run script
bash /home/tim/e63-coursework/hw6/scripts/splitAndSend2.sh
```

Count-buys

```
spark-submit --master local[4] /home/tim/e63-coursework/hw6/scripts/count-buys2.py
```

```
tim@ip-172-31-24-35:~/e63-coursework/hw6/scripts$ spark-submit \
--master local[4] /home/tim/e63-coursework/hw6/scripts/count-buys4.py
-------------------------------------------
Batch: 0
-------------------------------------------

[Stage 0:>                                                          (0 + 3) / 3]
[Stage 1:===>                                                     (12 + 4) / 200]
[Stage 1:=====>                                                   (21 + 4) / 200]
[Stage 1:=======>                                                 (26 + 4) / 200]
[Stage 1:=========>                                               (35 + 4) / 200]
[Stage 1:============>                                            (43 + 4) / 200]
[Stage 1:==============>                                          (53 + 4) / 200]
[Stage 1:==================>                                      (64 + 4) / 200]
[Stage 1:======================>                                  (75 + 5) / 200]
```

```
[Stage 1:======================>                                  (85 + 4) / 200]
[Stage 1:=========================>                               (97 + 4) / 200]
[Stage 1:============================>                          (110 + 4) / 200]
[Stage 1:==============================>                        (123 + 4) / 200]
[Stage 1:================================>                      (135 + 4) / 200]
[Stage 1:====================================>                  (149 + 4) / 200]
[Stage 1:======================================>                (159 + 4) / 200]
[Stage 1:========================================>              (171 + 4) / 200]
[Stage 1:==========================================>            (187 + 4) / 200]

+---+--------------------+-----+
|buy|              window|count|
+---+--------------------+-----+
|  S|[2016-03-22 20:25...|14871|
|  B|[2016-03-22 20:25...|15129|
+---+--------------------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------

[Stage 4:======================================>                  (2 + 1) / 3]
[Stage 5:============>                                           (46 + 4) / 200]
[Stage 5:================>                                       (59 + 4) / 200]
[Stage 5:====================>                                   (72 + 4) / 200]
[Stage 5:========================>                               (86 + 4) / 200]
[Stage 5:============================>                          (101 + 4) / 200]
[Stage 5:================================>                      (116 + 4) / 200]
[Stage 5:====================================>                  (134 + 4) / 200]
[Stage 5:========================================>              (151 + 4) / 200]
[Stage 5:============================================>          (163 + 4) / 200]
[Stage 5:================================================>      (179 + 4) / 200]
[Stage 5:==================================================> (193 + 5) / 200]

+---+--------------------+-----+
|buy|              window|count|
+---+--------------------+-----+
|  S|[2016-03-22 20:25...|27556|
|  B|[2016-03-22 20:25...|28012|
|  B|[2016-03-22 20:25...| 2236|
|  S|[2016-03-22 20:25...| 2196|
+---+--------------------+-----+

-------------------------------------------
Batch: 2
-------------------------------------------

[Stage 9:==========>                                             (37 + 4) / 200]
[Stage 9:==============>                                         (50 + 4) / 200]
[Stage 9:==================>                                     (67 + 4) / 200]
[Stage 9:========================>                               (83 + 4) / 200]
[Stage 9:============================>                          (98 + 4) / 200]
[Stage 9:================================>                      (110 + 4) / 200]
[Stage 9:==================================>                    (125 + 4) / 200]
```

```
[Stage 9:=======================================>                 (145 + 4) / 200]
[Stage 9:=============================================>           (162 + 4) / 200]
[Stage 9:================================================>        (181 + 4) / 200]

+---+--------------------+-----+
|buy|              window|count|
+---+--------------------+-----+
|  S|[2016-03-22 20:25...|27556|
|  B|[2016-03-22 20:25...|28012|
|  B|[2016-03-22 20:25...| 7201|
|  S|[2016-03-22 20:25...| 7231|
+---+--------------------+-----+
```

The above code shows, that the new adapted script is running perfectly in the structured streaming mode.

Problem 5 (20%)

Question: Examine provided Python program stateful_wordcount.py. Make it work as is. If there are errors on the code, fix them.

i) Stateful wordcount

The script *stateful-wordcount.py* counts words in UTF8 encoded, '\n' delimited text received from the nework every second. To run the stateful_wordcount first the netcat server has to be run.

```
nc -lk 9999
```

Note: The *stateful_wordcount.py* script has produces errors as " instead of # are used to comment the text. After fixing this, the code is running:

```
spark-submit --master local[4] \
/home/tim/e63-coursework/hw6/scripts/stateful_wordcount.py localhost 9999
```

Netcat server

```
tim@ip-172-31-24-35:~/e63-coursework/hw6/data$ nc -lk 9999
hello world
this is a test
talking to the world
an apple a day, keeps the doctor away
between you and me, britney is beautiful
```

Stateful wordcount

```
-------------------------------------------
Time: 2017-10-13 15:50:45
-------------------------------------------
(u'this', 1)
(u'a', 2)
(u'and', 1)
(u'test', 1)
(u'beautiful', 1)
```

```
(u'apple', 1)
(u'to', 1)
(u'me,', 1)
(u'talking', 1)
(u'world', 2)
...
```

The above output shows, that the stateful wordcount is working.

ii) Word count (a & b)

Question: Modify the code so that it outputs the number of words starting with letters a and b. Demonstrate that modified program work. You should provide several both positive and negative examples.

Next we're rewriting the the script. The code from the resulting *stateful_wordcount2.py* is below:

```
# Load libraries
from __future__ import print_function
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Exit condition
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: stateful_network_wordcount.py <hostname> <port>", file=sys.stderr)
        exit(-1)

    # Load Sparkcontext
    sc = SparkContext(appName="PythonStreamingStatefulNetworkWordCount")
    ssc = StreamingContext(sc, 3)
    ssc.checkpoint("checkpoint")

    # Define update function
    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

    # Load data
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

    # Map and filter data
    running_counts = lines.flatMap(lambda line: line.split(" "))\
                          .filter(lambda x: x.startswith("a") or x.startswith("b"))\
                          .map(lambda word: (word, 1))\
                          .updateStateByKey(updateFunc)

    # Print data
    running_counts.pprint()

    ssc.start()
    ssc.awaitTermination()
```

As the *netcat* server is still running we only have to run the *stateful_wordcount2.py* script from above.

```
spark-submit --master local[4] \
/home/tim/e63-coursework/hw6/scripts/stateful_wordcount2.py localhost 9999
```

Netcat server

```
tim@ip-172-31-24-35:~/e63-coursework/hw6/data$ nc -lk 9999
hello world
this is a test
talking to the world
an apple a day, keeps the doctor away
between you and me, britney is beautiful
```

Stateful wordcount

```
-------------------------------------------
Time: 2017-10-13 16:01:48
-------------------------------------------
(u'a', 2)
(u'and', 1)
(u'beautiful', 1)
(u'apple', 1)
(u'away', 1)
(u'britney', 1)
(u'an', 1)
(u'between', 1)
```

The above code shows, that all the words starting with an a or b are selected (positives). Words starting with a different letter are not selected (negatives).