**BIG DATA  
 & INTELLIGENT ANALYTICS**

**ASSIGNMENT 2**

**REPORT**



**Prepared by**

**Anirudha Deepak Bedre**

**Avikal Chhetri**

**(TEAM 3)**

**Index**

Table of Contents

**Introduction3**

**Kafka – Wordcount – Scala Implementation4**

**Kafka – Wordcount – Python Implementation5**

**Flume – Wordcount – Scala Implementation6**

**Flume – Wordcount – Scala Implementation7**

**Kinesis – Clickstream Implementation8**

**HDFS – Wordcount – Scala Implementation9**

**HDFS – Wordcount – Python Implementation10**

**MQTT – Hello World – Scala Implementation11**

**Twitter – Twitter Popular Tags – Scala Implementation12**

**ZeroMQ – Wordcount – Scala Implementation13**

**Lessons Learned15**

**Conclusion15**

**Introduction**

The following report consists of different streaming algorithms performed in Apache Spark on AWS EMR, namely:

Kafka-wordcount (Scala & Python)

Flume-wordcount (Scala & Python)

Kinesis –Clickstreamanalysis (Scala)

HDFS –wordcount (Scala & Python)

MQTT –Hello world (Scala)

Twitter –Twitter Popular Tags (Scala)

ZeroMQ-wordcount (Scala)

**Disclaimer:**

Some of the algorithms work in Spark 1.3.1 and some work in 1.4.1

We have mentioned the Spark version in which algorithms work in their respective documents.

**1: Kafka Wordcount**

Tested in

Spark version: 1.3.1

Kafka version: 2.11-0.8.2.1

Scala Implementation

**Start Zookeeper server**

bin/zookeeper-server-start.sh /home/hadoop/kafka\_2.11-0.8.2.1/kafka\_2.11-0.8.2.1/config/zookeeper.properties

**In a parallel terminal, Start Kafka Server**

bin/kafka-server-start.sh /home/hadoop/kafka\_2.11-0.8.2.1/kafka\_2.11-0.8.2.1/config/server.properties

**In a parallel terminal, Make Kafka Topic (for eg: bigdatatopic)**

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic bigdatatopic

**In a parallel terminal, Start Producer**

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic bigdatatopic

**In a parallel terminal, Start Consumer**

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic bigdatatopic --from-beginning

**In a parallel terminal, Run in spark directory**

bin/run-example org.apache.spark.examples.streaming.KafkaWordCount localhost:2181 my-consumer-group bigdatatopic

Python Implementation

Could implement this only in local system with spark version 1.4.0

The issue with EMR Cluster running Spark 1.4.1 was that Zookeeper server kept getting timed out.

**Start Zookeeper server**

bin/zookeeper-server-start.sh /home/hadoop/kafka\_2.11-0.8.2.1/kafka\_2.11-0.8.2.1/config/zookeeper.properties

**In a parallel terminal, Start Kafka Server**

bin/kafka-server-start.sh /home/hadoop/kafka\_2.11-0.8.2.1/kafka\_2.11-0.8.2.1/config/server.properties

**In a parallel terminal, Make Kafka Topic (for eg: bigdatatopic)**

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic bigdatatopic

**In a parallel terminal, Start Producer**

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic bigdatatopic

**In a parallel terminal, Start Consumer**

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic bigdatatopic --from-beginning

**In a parallel terminal, Run in spark directory**

bin/spark-submit --packages org.apache.spark:spark-streaming-kafka\_2.10:1.3.1 --jars $(echo /home/hadoop/spark/lib/\*.jar | tr ' ' ',') examples/src/main/python/streaming/kafka\_wordcount.py localhost:2181 bigdatatopic

**2: Flume Wordcount**

Tested in

Spark version: 1.3.1

Flume version: 1.6.0

We could not completely executed Flume Event Count in scala, as flume word count is not present.

Scala Implementation

**First we have to create a configuration file by the name avro\_spark.conf in flume/conf/ directory**

a1.sources = r1

a1.channels = c1

a1.sources.r1.type = avro

a1.sources.r1.channels = c1

a1.sources.r1.bind = 172.31.37.177

a1.sources.r1.port = 4141

a1.sinks = k1

a1.sinks.k1.type = avro

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

a1.sinks.k1.hostname = 172.31.37.177

a1.sinks.k1.port = 6666

a1.sources = r1

a1.sinks = spark

a1.channels = c1

**The configurations mentioned in the above file are not completely correct. We tried various configurations but did not succeed.**

**In flume/bin directory execute the following command to configure the above properties**

flume-ng agent -c . -f conf/avro\_spark.conf -n a1 Start Spark-streaming

**In spark directory, execute the following command to run the Flume event listener**

bin/run-example org.apache.spark.examples.streaming.FlumeEventCount 172.31.37.177 6666

**In Parallel terminal, create a wordcount.txt file and execute the following command through an avro client in flume/bin directory**

flume-ng avro-client -c . -H 172.31.37.177 -p 4141 -F wordcount.txt

Python Implementation

**First we have to create a configuration file by the name avro\_spark.conf in flume/conf/ directory**

a1.sources = r1

a1.channels = c1

a1.sources.r1.type = avro

a1.sources.r1.channels = c1

a1.sources.r1.bind = 172.31.37.177

a1.sources.r1.port = 4141

a1.sinks = k1

a1.sinks.k1.type = avro

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

a1.sinks.k1.hostname = 172.31.37.177

a1.sinks.k1.port = 6666

a1.sources = r1

a1.sinks = spark

a1.channels = c1

**The configurations mentioned in the above file are not completely correct. We tried various configurations but did not succeed.**

**In flume/bin directory execute the following command to configure the above properties**

flume-ng agent -c . -f conf/avro\_spark.conf -n a1 Start Spark-streaming

**In spark directory, execute the following command to run the Flume event listener**

bin/spark-submit --jars external/flume-assembly/target/scala-\*/ spark-streaming-flume-assembly-\*.jar examples/src/main/python/streaming/flume\_wordcount.py localhost 172.31.37.177

**In Parallel terminal, create a wordcount.txt file and execute the following command through an avro client in flume/bin directory**

flume-ng avro-client -c . -H 172.31.37.177 -p 4141 -F wordcount.txt

**3: Kinesis Clickstream Analysis**

Tested in

Spark version: 1.4.1

Scala Implementation

**Run the generator**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewGenerator 44444 10

In a parallel terminal , To process the generated stream

**For PageCounts**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewStream pageCounts localhost 44444

**For Sliding Page Counts**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewStream slidingPageCounts localhost 44444

**For Error Rate Per ZipCode**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewStream errorRatePerZipCode localhost 44444

**For Active User Count**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewStream activeUserCount localhost 44444

**For Popular Users Seen**

bin/run-example org.apache.spark.examples.streaming.clickstream.PageViewStream popularUsersSeen localhost 44444

**4: HDFS Wordcount**

Tested in

Spark version: 1.4.1

Scala Implementation

**Create wordcount.txt file in local AWS directory**

**To edit the textfile**

sudo vi wordcount.txt

**Create a directory in HDFS to put the file later on:**

hadoop fs -mkdir /user/ani

hadoop fs -mkdir /user/ani/localdir

**Run the Scala script**

bin/run-example org.apache.spark.examples.streaming.HdfsWordCount /user/ani/localdir

**In parallel terminal, move the wordcount.txt file from local AWS directory to HDFS directory created above, using the following command**

hadoop fs -put wordcount.txt /user/ani/localdir

Python Implementation

**Create wordcount.txt file in local AWS directory**

**To edit the textfile**

sudo vi wordcount.txt

**Create a directory in HDFS to put the file later on:**

hadoop fs -mkdir /user/ani

hadoop fs -mkdir /user/ani/localdir

**Run the Python script**

bin/spark-submit examples/src/main/python/streaming/hdfs\_wordcount.py /user/ani/localdir

**In parallel terminal, move the wordcount.txt file from local AWS directory to HDFS directory created** **above, using the following command**

hadoop fs -put wordcount.txt /user/ani/localdir

**5: MQTT – Hello World**

Tested in

Spark version: 1.3.1

MQTT version: 1.4.2

Scala Implementation

**Initial Setup:**

**Copy mosquitto-1.4.2.tar.gz to AWS**

**Unzip the file using the command:**

tar -vxzf mosquitto-1.4.2.tar.gz

**Navigate to mosquitto-1.4.2 directory**

cd mosquitto-1.4.2

**Install cmake to build mosquito using the following commands**

sudo yum install cmake

cmake .

sudo make install

**Run the following command to start mosquitto broker**

mosquitto

**In a parallel terminal, run the publisher**

bin/run-example org.apache.spark.examples.streaming.MQTTPublisher tcp://localhost:1883 foo

**In a parallel terminal, run the consumer**

bin/run-example org.apache.spark.examples.streaming.MQTTWordCount tcp://localhost:1883 foo

**6: Twitter- Twitter Popular Tags**

Tested in

Spark version: 1.3.1

Scala Implementation

**Create a twitter application** [here](https://apps.twitter.com) **(to get access key and token)**

**Make a twitter4j.properties file and store in Spark directory**

**twitter4j.properties file should contain :**

oauth.consumerKey = RLDuPu9vwZ2ZZtLuFk079NA3Z

oauth.consumerSecret = hEs7rCWkZshBokJrAedZ6ED9iBA5hZqGIKj123OWyUl7HLrdKj

oauth.accessToken = 67358709-QnsxyrZLQBlwjxBdllD5mBRcDsEAUZOQ9NDUJrFiC

oauth.accessTokenSecret = gKpiYnrgo4EmefLw1NAUua70DaP15Ar5cL0kYS9ksPpuC

**Run the scala script by passing the authentication keys as arguments**

bin/run-example org.apache.spark.examples.streaming.TwitterPopularTags RLDuPu9vwZ2ZZtLuFk079NA3Z hEs7rCWkZshBokJrAedZ6ED9iBA5hZqGIKj123OWyUl7HLrdKj 67358709-QnsxyrZLQBlwjxBdllD5mBRcDsEAUZOQ9NDUJrFiC gKpiYnrgo4EmefLw1NAUua70DaP15Ar5cL0kYS9ksPpuC

**7: ZeroMQ- Wordcount**

Tested in

Spark version: 1.3.1

ZeroMQ version: 2.2.0

Scala Implementation

Install the following prerequisite packages

sudo yum install libtool

sudo yum install autoconf

sudo yum install automake

sudo yum install gcc-c++

sudo yum install libuuid-devel

Copy the following ZeroMQ packages

zeromq-2.2.0

jzmq-master

libzmq-master

Navigate to zeromq-2.2.0 directory

./autogen.sh

./configure --prefix=/home/hadoop/

make

make install

sudo ldconfig -v

Navigate to libzmq-master directory

./autogen.sh

./configure --without-libsodium --prefix=/home/hadoop/

make

make install

sudo ldconfig -v

Navigate to jzmq-master

./autogen.sh

./configure --without-libsodium --prefix=/home/hadoop/

We are getting error here quoting unable to find zmq file. We tried adding the library path to refer to library files but still didn’t work.

Rest of the steps if this works, are as follows:

make

make install

sudo ldconfig -v

Run the following commands in spark directory

bin/run-example org.apache.spark.examples.streaming.SimpleZeroMQPublisher tcp://[127.0.1.1:1234](http://127.0.1.1:1234/) foo.bar

bin/run-example org.apache.spark.examples.streaming.ZeroMQWordCount tcp://[127.0.1.1:1234](http://127.0.1.1:1234/) foo

**Lessons learnt and challenges faced**

Apache Spark 1.4.1 did not support most of the streaming algorithms as a lot of the jars (although present) were not referenced during compile time.

**Conclusion**

Thus Spark streaming algorithms integrated with different streaming applications were explored successfully.