//W251 Final Project Spark Processing

//Log into machine

ssh root@158.85.217.74

T3jwTq62

for i in 0.0.0.0 spark1dnm spark2dnm spark3dnm; do ssh $i; done

cd /sandisk1/darknetmarket/silkroad2-forums/final

//Start shell with Cassandra connector

$SPARK\_HOME/bin/spark-shell --packages datastax:spark-cassandra-connector:1.5.0-s\_2.10

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

import com.datastax.spark.connector.\_

val conf = new SparkConf(true).set("spark.cassandra.connection.host", "158.85.217.74")

val sc = new SparkContext("local", "test", conf)

val topics\_db = sc.cassandraTable("dnm","topics")

// original posts – 26,392

val topics\_db\_original\_posts = topics\_db.filter(row => row.getString("responder\_numposts") == "nan").map{ row => (row.getString("date"), row.getString("author"), row.getString("response\_date"), row.getString("responder"), row.getString("topic") )}

// response posts – 1,368,211

val topics\_db\_response\_posts = topics\_db.filter(row => row.getString("responder\_numposts") != "nan").map{ row => (row.getString("date"), row.getString("author"), row.getString("response\_date"), row.getString("responder"), row.getString("topic") )}

// filter by date

// get count of the number of posts per date

val counts\_original = topics\_db\_original\_posts.map(row => row.\_1).map(\_.split(" ")(0))

counts\_original.saveAsTextFile("counts\_original")

val counts\_response = topics\_db\_response\_posts.map(row => row.\_1).map(\_.split(" ")(0))

counts\_response.saveAsTextFile("counts\_response")

exit

//start standard shell

$SPARK\_HOME/bin/spark-shell

val original\_counts = sc.textFile("counts\_original/part-00000")

val new\_original\_counts = original\_counts.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_original\_counts\_rdd = sc.parallelize(new\_original\_counts)

new\_original\_counts\_rdd.saveAsTextFile("counts\_original\_reduced")

val response\_counts = sc.textFile("counts\_response/part-00000")

val new\_response\_counts = response\_counts.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_response\_counts\_rdd = sc.parallelize(new\_response\_counts)

new\_response\_counts\_rdd.saveAsTextFile("counts\_response\_reduced")

// filter by number of active users per date

//in Cassandra connector shell

val counts\_users\_original1 = topics\_db\_original\_posts.map(row => (row.\_1,row.\_2)).map(row => (row.\_1.split(" ")(0),row.\_2))

counts\_users\_original1.saveAsTextFile("counts\_users\_original")

val counts\_users\_response = topics\_db\_response\_posts.map(row => (row.\_1,row.\_4)).map(row => (row.\_1.split(" ")(0),row.\_2))

counts\_users\_response.saveAsTextFile("counts\_users\_response")

//in standard shell

val original\_users\_counts = sc.textFile("counts-users-original/part-00000 ")

val new\_original\_users\_counts = original\_users\_counts.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_original\_users\_counts\_rdd = sc.parallelize(new\_original\_users\_counts)

val new\_original\_users\_counts\_by\_user = new\_original\_users\_counts\_rdd.map(\_.\_1.split(",")(0)).map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val original\_users\_perday\_rdd = sc.parallelize(new\_original\_users\_counts\_by\_user)

original\_users\_perday\_rdd.saveAsTextFile("counts\_original\_users\_reduced")

val response\_users\_counts = sc.textFile("counts-users-response/part-00000 ")

val new\_response\_users\_counts = response\_users\_counts.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_response\_users\_counts\_rdd = sc.parallelize(new\_response\_users\_counts)

val new\_response\_users\_counts\_by\_user = new\_response\_users\_counts\_rdd.map(\_.\_1.split(",")(0)).map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val response\_users\_perday\_rdd = sc.parallelize(new\_response\_users\_counts\_by\_user)

response\_users\_perday\_rdd.saveAsTextFile("counts\_response\_users\_reduced")

//top topics per day

//in Cassandra Connector shell

val words\_day\_ori = topics\_db\_original\_posts.map(row => (row.\_1,row.\_5)).map(row => (row.\_1.split(" ")(0),row.\_2))

words\_day\_ori.saveAsTextFile("words\_per\_day")

val topics\_day\_res = topics\_db\_response\_posts.map(row => (row.\_1,row.\_5)).map(row => (row.\_1.split(" ")(0),row.\_2))

topics\_day\_res.saveAsTextFile("res\_topics\_per\_day")

//in standard shell

val words\_day\_ori = sc.textFile("words\_per\_day/part-00000")

val new\_words\_day\_ori = words\_day\_ori.map(\_.replace("(","")).map(\_.replace(")","")).map(row => (row.substring(0,10),row.substring(11,row.length))).map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val topics\_day\_ori\_rdd = sc.parallelize(new\_words\_day\_ori)

topics\_day\_ori\_rdd.saveAsTextFile("topics\_per\_day\_ori")

val topics\_day\_res = sc.textFile("part-00000")

val new\_topics\_day\_res = topics\_day\_res.map(\_.replace("(","")).map(\_.replace(")","")).map(row => (row.substring(0,10),row.substring(11,row.length))).map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val topics\_day\_res\_rdd = sc.parallelize(new\_topics\_day\_res)

topics\_day\_res\_rdd.saveAsTextFile("topics\_per\_day\_res")

// Topics per users

$SPARK\_HOME/bin/spark-shell --packages datastax:spark-cassandra-connector:1.5.0-s\_2.10

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

import com.datastax.spark.connector.\_

val conf = new SparkConf(true).set("spark.cassandra.connection.host", "158.85.217.74")

val sc = new SparkContext("local", "test", conf)

val topics\_db = sc.cassandraTable("dnm","topics")

val topics\_db\_original\_posts = topics\_db.filter(row => row.getString("responder\_numposts") == "nan").map{ row => (row.getString("date"), row.getString("author"), row.getString("response\_date"), row.getString("responder"), row.getString("topic") )}

val topics\_db\_response\_posts = topics\_db.filter(row => row.getString("responder\_numposts") != "nan").map{ row => (row.getString("date"), row.getString("author"), row.getString("response\_date"), row.getString("responder"), row.getString("topic") )}

val topics\_user\_ori = topics\_db\_original\_posts.map(row => (row.\_2,row.\_5))

topics\_user\_ori.saveAsTextFile("topics\_user\_count\_ori")

val topics\_user\_res = topics\_db\_response\_posts.map(row => (row.\_4,row.\_5))

topics\_user\_res.saveAsTextFile("topics\_user\_count\_res")

$SPARK\_HOME/bin/spark-shell

val topics\_user\_ori = sc.textFile("topics\_user\_count\_ori/part-00000")

val new\_topics\_user\_ori = topics\_user\_ori.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_topics\_user\_ori\_rdd = sc.parallelize(new\_topics\_user\_ori)

new\_topics\_user\_ori\_rdd.saveAsTextFile("topics\_user\_count\_ori\_reduced")

val topics\_user\_res = sc.textFile("topics\_user\_count\_res/part-00000-res")

val new\_topics\_user\_res = topics\_user\_res.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val new\_topics\_user\_res\_rdd = sc.parallelize(new\_topics\_user\_res)

new\_topics\_user\_res\_rdd.saveAsTextFile("topics\_user\_count\_res\_reduced")

// For network analysis

$SPARK\_HOME/bin/spark-shell --packages datastax:spark-cassandra-connector:1.5.0-s\_2.10

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

import com.datastax.spark.connector.\_

val conf = new SparkConf(true).set("spark.cassandra.connection.host", "158.85.217.74")

val sc = new SparkContext("local", "test", conf)

val topics\_db = sc.cassandraTable("dnm","topics")

val topics\_db\_response\_posts = topics\_db.filter(row => row.getString("responder\_numposts") != "nan").map{ row => (row.getString("date"), row.getString("author"), row.getString("response\_date"), row.getString("responder"), row.getString("topic") )}

# find responder to author

val responder\_to\_author = topics\_db\_response\_posts.map(row => (row.\_4,row.\_2))

responder\_to\_author.saveAsTextFile("responder\_to\_author")

# unique usernames

val author\_names = topics\_db\_response\_posts.map(row => row.\_2)

author\_names.saveAsTextFile("names\_author")

val responder\_names = topics\_db\_response\_posts.map(row => row.\_4)

responder\_names.saveAsTextFile("names\_responder")

$SPARK\_HOME/bin/spark-shell

// 616017 edges

val responder\_to\_author = sc.textFile("part-00000-r2a")

val edges = responder\_to\_author.map(word => (word, 1)).reduceByKey(\_+\_).sortByKey(true).collect()

val edges\_rdd = sc.parallelize(edges)

edges\_rdd.saveAsTextFile("edges\_reduced")

// 52650 nodes

val author\_names = sc.textFile("part-00000-name-a")

val responder\_names = sc.textFile("part-00000-name-r")

val names\_rdd = author\_names.union(responder\_names)

val names\_new = names\_rdd.distinct()

names\_new.saveAsTextFile("nodes\_reduced")

// Graphx

//cleaning

val r2a1 = sc.textFile("edges\_reduced/part-00000")

val r2a2 = sc.textFile("edges\_reduced/part-00001")

val r2a3 = sc.textFile("edges\_reduced/part-00002")

val r2a4 = sc.textFile("edges\_reduced/part-00003")

val edges\_file = r2a1.union(r2a2).union(r2a3).union(r2a4)

val edges\_rdd = edges\_file.map(row => row.replace("(","")).map(row => row.replace(")","")).map(row => (row.split(","))).collect()

val edges\_new\_rdd = sc.parallelize(edges\_rdd)

val edges\_new = edges\_new\_rdd.map(row => (row(0),row(1),row(row.length-1).toInt))

edges\_new.coalesce(1).saveAsTextFile("neo4j\_edges")

// new edges = 562445, after filter

val edges\_final = edges\_new.filter(row => row.\_1 != "nan").filter(row => row.\_2 != "nan").filter(row => row.\_1 != row.\_2)

edges\_final.coalesce(1).saveAsTextFile("neo4j\_edges\_final")

val node1 = sc.textFile("nodes\_reduced/part-00000")

val node2 = sc.textFile("nodes\_reduced/part-00001")

val node3 = sc.textFile("nodes\_reduced/part-00002")

val node4 = sc.textFile("nodes\_reduced/part-00003")

val nodes\_file = node1.union(node2).union(node3).union(node4)

nodes\_file.coalesce(1).saveAsTextFile("neo4j\_nodes")

//from <http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html>

//from <http://www.sparktutorials.net/analyzing-flight-data:-a-gentle-introduction-to-graphx-in-spark>

//from <https://www.mapr.com/blog/how-get-started-using-apache-spark-graphx-scala>

// from https://www.mapr.com/blog/how-get-started-using-apache-spark-graphx-scala

import org.apache.spark.graphx.\_

import org.apache.spark.rdd.RDD

import scala.util.MurmurHash

val vertices: RDD[(VertexId, String)] = nodes\_file.map(x => (MurmurHash.stringHash(x), x))

val edges = edges\_final.map(x => ((MurmurHash.stringHash(x.\_1.toString),MurmurHash.stringHash(x.\_2.toString)), x.\_3)).reduceByKey(\_+\_).map(x => Edge(x.\_1.\_1, x.\_1.\_2,x.\_2))

// Defining a default vertex called nowhere

val noone = "noone"

// define the graph

val graph = Graph(vertices,edges,noone)

//1. How many users are there?

// How many users?

val numusers = graph.numVertices

// numusers: Long = 52664

//2. How many relationships are there?

// How many relationships?

val numrelationships = graph.numEdges

// numrelationships: Long = 562116

//3. Sort and print out the most frequent/strongest relationships

// print out most frequent/strongest relationships

graph.triplets.sortBy(\_.attr, ascending=false).map(triplet => "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " + triplet.dstAttr + ".").collect.take(20).foreach(println)

Distance 1217 from noone to noone.

Distance 162 from giancarlo to Merde222.

Distance 146 from ScrewsLoose to Merde222.

Distance 140 from mary666 to ChemCat.

Distance 133 from mary666 to CaptainWhiteBeard.

Distance 122 from murderface2012 to Merde222.

Distance 116 from ChemCat to mary666.

Distance 114 from calcium345 to CaptainWhiteBeard.

Distance 114 from doctorwhat to ChemCat.

Distance 113 from murderface2012 to ACE.

Distance 100 from murderface2012 to smity1020.

Distance 99 from smity1020 to CaptainWhiteBeard.

Distance 99 from TheSlyFox to diax.

Distance 98 from Merde222 to giancarlo.

Distance 98 from murderface2012 to ChemCat.

Distance 97 from BoxofShapes to ChemCat.

Distance 93 from deathowl1990 to CaptainWhiteBeard.

Distance 92 from PillfirePharmacy to mary666.

Distance 91 from chemicals\_spain to Merde222.

Distance 88 from mary666 to smity1020.

//4. Sort and print out the least frequent/weakest relationships

// print out least frequent/weakest relationships

graph.triplets.sortBy(\_.attr).map(triplet => "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " + triplet.dstAttr + ".").take(20).foreach(println)

Distance 1 from Holland\_SR to frogwithADD.

Distance 1 from fotwentee to Treignrex.

Distance 1 from fotwentee to one2bcurious.

Distance 1 from fotwentee to northsouth.

Distance 1 from robbiefowler23 to googleyed1.

Distance 1 from VANQUISH4777 to thecatisback.

Distance 1 from VANQUISH4777 to SandStorm.

Distance 1 from VANQUISH4777 to TheDemiGod.

Distance 1 from VANQUISH4777 to SourDiesel.

Distance 1 from VANQUISH4777 to chakalaka.

Distance 1 from VANQUISH4777 to loqum.

Distance 1 from Bonfils1 to Snoopish.

Distance 1 from Bonfils1 to DiddleMyThoughts.

Distance 1 from Throwaway7839 to jdddd.

Distance 1 from CannaMed to uniwiz.

Distance 1 from stickyickyicky23 to opopko.

Distance 1 from semilanceata to fancyfutwork.

Distance 1 from moshe to 124ab87c.

Distance 1 from moshe to baltazar666.

Distance 1 from tadeus99 to baller69.

//5. what user has the most in degrees or unique users into it?

graph.inDegrees.join(vertices).sortBy(\_.\_2.\_1, ascending=false).take(20).foreach(println)

(546159859,(1726,ChemCat))

(-1477396411,(917,DoctorClu))

(254234108,(805,murderface2012))

(208702416,(799,Limetless))

(-812369126,(760,Hiniguel))

(159345784,(719,giancarlo))

(-1221072419,(708,CaptainWhiteBeard))

(820108162,(701,HonoluluExpress))

(-127955699,(682,moonbear))

(-187502943,(597,mary666))

(1580938761,(594,FatherTed))

(886968989,(567,smity1020))

(944669147,(556,twatWaffle))

(-206116113,(530,Merde222))

(-369167194,(525,ScrewsLoose))

(-419080413,(520,MDMA80))

(-181474775,(515,mito))

(-2144365289,(504,PerfectScans))

(1633118482,(497,TragicallyHip))

(487862362,(491,doctorwhat))

//6. what user has the most out degrees or unique users out of it?

graph.outDegrees.join(vertices).sortBy(\_.\_2.\_1, ascending=false).take(20).foreach(println)

(546159859,(2183,ChemCat))

(209060421,(1966,Tang))

(208702416,(1192,Limetless))

(-319647961,(1189,scout))

(2092013915,(1084,TheSlyFox))

(1386904961,(986,fallingsnow))

(-1477396411,(938,DoctorClu))

(-227248955,(932,cryngie))

(1998156406,(865,Libertas))

(254234108,(824,murderface2012))

(-226531436,(821,Yoda))

(1227346810,(776,Nightcrawler))

(1580938761,(768,FatherTed))

(1668896828,(748,astor))

(-1987070084,(746,SelfSovereignty))

(-1221072419,(723,CaptainWhiteBeard))

(-367781719,(680,Cornelius23))

(820108162,(677,HonoluluExpress))

(1928209267,(672,dirtybiscuitzz718))

(-1153633307,(659,BigTenInch\_\_Record))

//Page Rank

val ranks = graph.pageRank(0.0001).vertices

//7. So what are our most important airports!?

val ranksUsers = ranks.join(vertices).sortBy(\_.\_2.\_1, ascending=false).map(\_.\_2.\_2)

ranksUsers.take(20).foreach(println)

scala> ranksUsers.take(20).foreach(println)

ChemCat

Limetless

DoctorClu

HonoluluExpress

murderface2012

Hiniguel

CaptainWhiteBeard

giancarlo

schnitzel\_karl

mito

moonbear

fallingsnow

Bungee54

Sarge

Tang

PerfectScans

FatherTed

Tessellated

purplelotus

AnonymousAddict

//8. Take the Top 10 most important and find their Top 5 most popular topics

val ori1 = sc.textFile("topics\_user\_count\_ori\_reduced/part-00000")

val ori2 = sc.textFile("topics\_user\_count\_ori\_reduced/part-00001")

val ori3 = sc.textFile("topics\_user\_count\_ori\_reduced/part-00002")

val ori4 = sc.textFile("topics\_user\_count\_ori\_reduced/part-00003")

val res1 = sc.textFile("topics\_user\_count\_res\_reduced/part-00000")

val res2 = sc.textFile("topics\_user\_count\_res\_reduced/part-00001")

val res3 = sc.textFile("topics\_user\_count\_res\_reduced/part-00002")

val res4 = sc.textFile("topics\_user\_count\_res\_reduced/part-00003")

val topics\_file = ori1.union(ori2).union(ori3).union(ori4).union(res1).union(res2).union(res3).union(res4)

//filter by user

val topics\_filter = topics\_file.map(row => row.replace("(","")).map(row => row.replace(")","")).map(row => (row.split(","))).collect()

val topics\_rdd = sc.parallelize(topics\_filter)

val topics\_user = topics\_rdd.filter(row => row(0) == "ChemCat").map(row => (row(row.length-1).toInt,row(1))).sortByKey(false).take(10).foreach(println)

(484,Re: Official Spare Coins Thread)

(309,Re: Official Spare Coins Thread READ OP BEFORE YOU REQUEST!! )

(304,Re: \*\*Spam to 50 &amp; Get out of Noobville\*\*)

(264,Re: READ THE 1ST PAGE!!!Official Spare Coins Thread READ OP BEFORE YOU REQUEST!! )

(209,Re: Official Spare Coins Thread )

(194,Re: Are you Paralyzed by PGP? Fear no more! Join PGP Club :)

(119,Re: The Green Camel Night Club)

(70,Re: Newbie PGP Club)

(60,Re: +++KARMA)

(53,Re: NEWBIES: Guide for learning PGP)

val topics\_user = topics\_rdd.filter(row => row(0) == "Limetless").map(row => (row(row.length-1).toInt,row(1))).sortByKey(false).take(10).foreach(println)

(111,Re: MEPHEDRONE VENDORS!)

(50,Re: READ FUCKING FIRST POST! YOUR NAME MUST BE IN IT - The Verbal Diarrhea Thread)

(37,Re: Chat more shit)

(17,Re: Limetless reviews)

(12,Re: + KARMA)

(10,Re: What music are you listening to right now?)

(7,Re: Last one to reply wins!)

(5,Re: Anyone recieve from SuperTrips?)

(5,Re: Get Vouched or Get Fucked - The Verbal Diarrhea Thread)

(5,Re: My account was hacked AGAIN by a MOD I suspect!!!!)

val topics\_user = topics\_rdd.filter(row => row(0) == "DoctorClu").map(row => (row(row.length-1).toInt,row(1))).sortByKey(false).take(10).foreach(println)

(67,Re: Vendor Verification for Round Table Access)

(61,Re: Official Spare Coins Thread)

(53,Re: +++KARMA)

(34,Re: The Green Camel Night Club)

(31,Re: Contingency Plan)

(25,Re: X)

(21,Re: StExo was behind the attack. Defcon did his best to defend the place. )

(14,Re: Buyer Blacklist)

(14,Re: Full Disclosure)

(14,Re: Inigo)

val topics\_user = topics\_rdd.filter(row => row(0) == "HonoluluExpress").map(row => (row(row.length-1).toInt,row(1))).sortByKey(false).take(10).foreach(println)

(88,Re: +++KARMA)

(48,Re: Official Spare Coins Thread READ OP BEFORE YOU REQUEST!! )

(45,Re: Official Spare Coins Thread)

(35,Re: The Green Camel Night Club)

(23,Re: I'm Here To Help The Newbies! FAQ Now Included In The OP!)

(20,Re: Official Help The Newbies Thread! FAQ Is Included In The OP! Updated 1/30/2014)

(19,Re: I'm Here To Help The Newbies!)

(14,Re: Unofficial List Of Vendors To FE With! Updated Frequently!)

(12,Re: Official Spare Coins Thread READ OP BEFORE YOU REQUEST!!)

(12,Re: Relaunch Timeline + Important Reminders UPDATE 2)

val topics\_user = topics\_rdd.filter(row => row(0) == "murderface2012").map(row => (row(row.length-1).toInt,row(1))).sortByKey(false).take(10).foreach(println)

(596,Re: The Scurvy Crew - Reviews and AWESOMENES Home of the finest hash)

(578,Re: ~OFFICIAL SilkRoad 2.0 Cocaine Thread -- First post modified with vendors~)

(343,Re: Official Spare Coins Thread)

(233,Re: Official Spare Coins Thread READ OP BEFORE YOU REQUEST!! )

(205,Re: OFFICIAL SilkRoad 2.0 Cocaine Thread)

(164,Re: Chemicals\_Spain review thread)

(156,Re: +++KARMA)

(155,Re: READ THE 1ST PAGE!!!Official Spare Coins Thread READ OP BEFORE YOU REQUEST!! )

(132,Re: The Green Camel Night Club)

(95,Re: ~OFFICIAL SilkRoad 2.0 Cocaine Thread -- No current vendors listed)