

Strata+ Hadoop WORLD

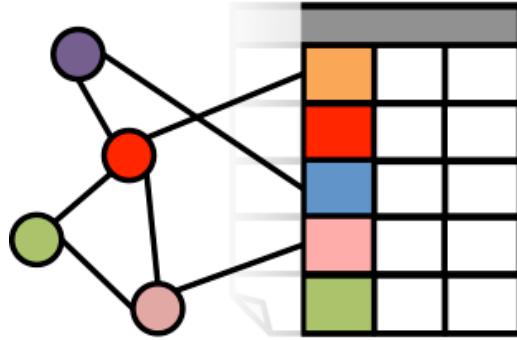
PRESNTED BY

O'REILLY®

cloudera®

strataconf.com

#StrataHadoop



GraphX

1. Intro
2. APIs
3. Hands-on

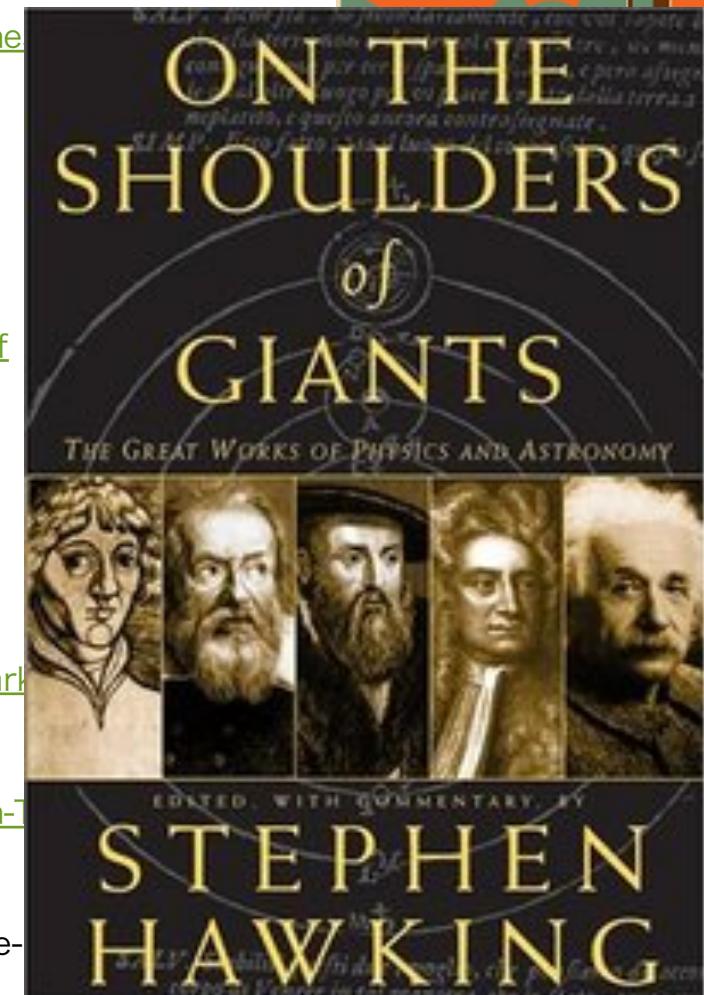
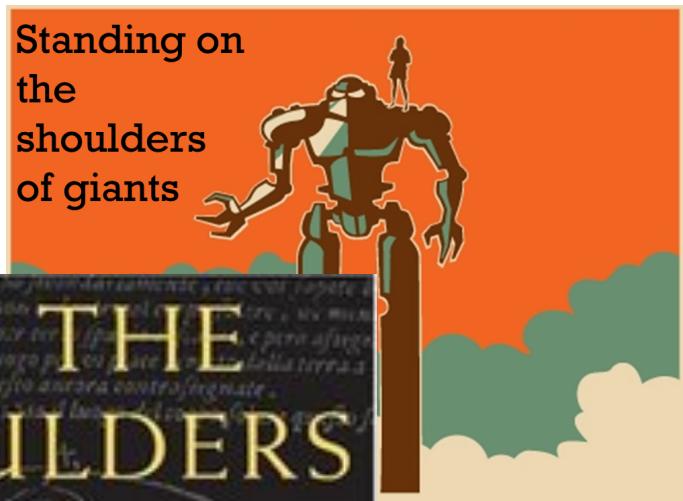
KRISHNA SANKAR

[HTTPS://WWW.LINKEDIN.COM/IN/KSANKAR](https://www.linkedin.com/in/ksankar)

MARCH 29, 2016

THANKS TO THE GIANTS WHOSE WORK HELPED TO PREPARE THIS TUTORIAL

1. [Paco Nathan, Scala Days 2015, https://www.youtube.com/watch?v=P_V71n-gtDs](https://www.youtube.com/watch?v=P_V71n-gtDs)
2. Big Data Analytics with Spark-Apress <http://www.amazon.com/Big-Data-Analytics-Spark-Practitioners/dp/1484209656>
3. Apache Spark Graph Processing-Packt <https://www.packtpub.com/big-data-and-business-intelligence/apache-spark-graph-processing>
4. Spark GraphX in Action - Manning
5. <http://hortonworks.com/blog/introduction-to-data-science-with-apache-spark/>
6. <http://stanford.edu/~rezab/nips2014workshop/slides/ankur.pdf>
7. Mining Massive Datasets book v2 <http://infolab.stanford.edu/~ullman/mmds/ch10.pdf>
- <http://web.stanford.edu/class/cs246/handouts.html>
8. <http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm>
9. <http://kukuruku.co/hub/algorithms/social-network-analysis-spark-graphx>
10. Zeppelin Setup
 - <http://sparktutorials.net/setup-your-zeppelin-notebook-for-data-science-in-apache-spark/>
11. Data
 - http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time
 - <http://openflights.org/data.html>
12. <https://adventuresincareerdevelopment.files.wordpress.com/2015/04/standing-on-the-shoulders-of-giants.png>



GraphX - Graph Applications

- Many applications from social network to understanding collaboration, diseases, routing logistics and others
- Came across two interesting applications, as I was preparing the slides!!

1) Project effectiveness by social network analysis on the projects' collaboration structures :

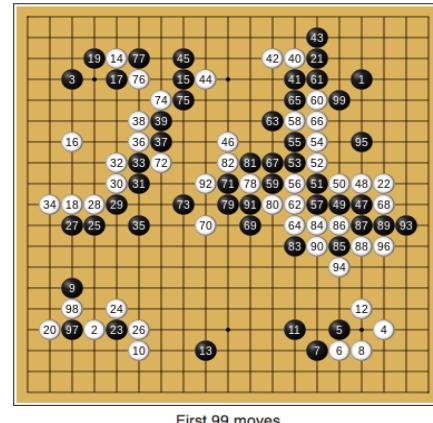
"EC is interested in the impact generated by those projects ... analyzing this latent impact of TEL projects by applying social network analysis on the projects' collaboration structures ... TEL projects funded under FP6, FP7 and eContentplus, and identifies central projects and strong, sustained organizational collaboration ties."

2) Weak ties in pathogen transmission : "structural motif... Giraffe social networks are characterized by social cliques in which individuals associate more with members of their own social clique than with those outside their clique ... Individuals involved in weak, between-clique social interactions are hypothesized to serve as bridges by which an infection may enter a clique and, hence, may experience higher infection risk ... individuals who engaged in more between-clique associations, that is, those with multiple weak ties, were more likely to be infected with gastrointestinal helminth parasites ... "



GraphX ...

- Graph processing at Scale - “Graph Parallel System”
- Has a rich
 - Computational Model
 - Edges, Vertices, Property Graph
 - Bipartite & tripartite graphs (Users-Tags-Web pages)
 - Algorithms (PageRank,...)
- Current focus on computation than query
- APIs include :
 - Attribute Transformers,
 - Structure transformers,
 - Connection Mining & Algorithms
- GraphFrames – interesting development
- Exercises (lots of interesting graph data)
 - Airline data, co-occurrence & co-citation from papers
 - The AlphaGo Community – ReTweet network
 - Wikipedia Page Rank Analysis using GraphX



Graphx Paper : <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-gonzalez.pdf>

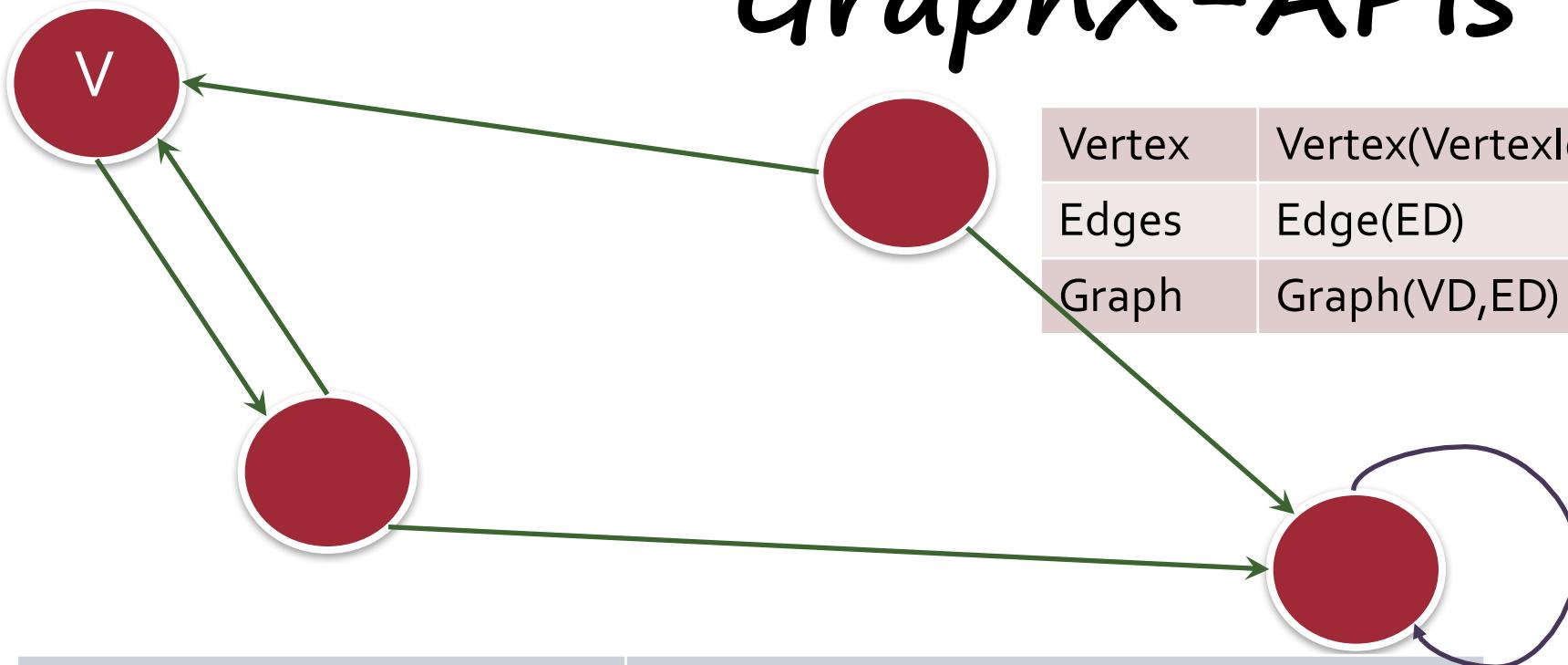
Pragel paper : http://kowshik.github.io/JPregel/pregeg_paper.pdf

Scala, python support <https://issues.apache.org/jira/browse/SPARK-3789>

Java API for Graphx <https://issues.apache.org/jira/browse/SPARK-3665>

LDA <https://issues.apache.org/jira/browse/SPARK-1405>

GraphX-APIs



Vertex	Vertex(VertexId,VD)	VD can be any object
Edges	Edge(ED)	ED can be any object
Graph	Graph(VD,ED)	

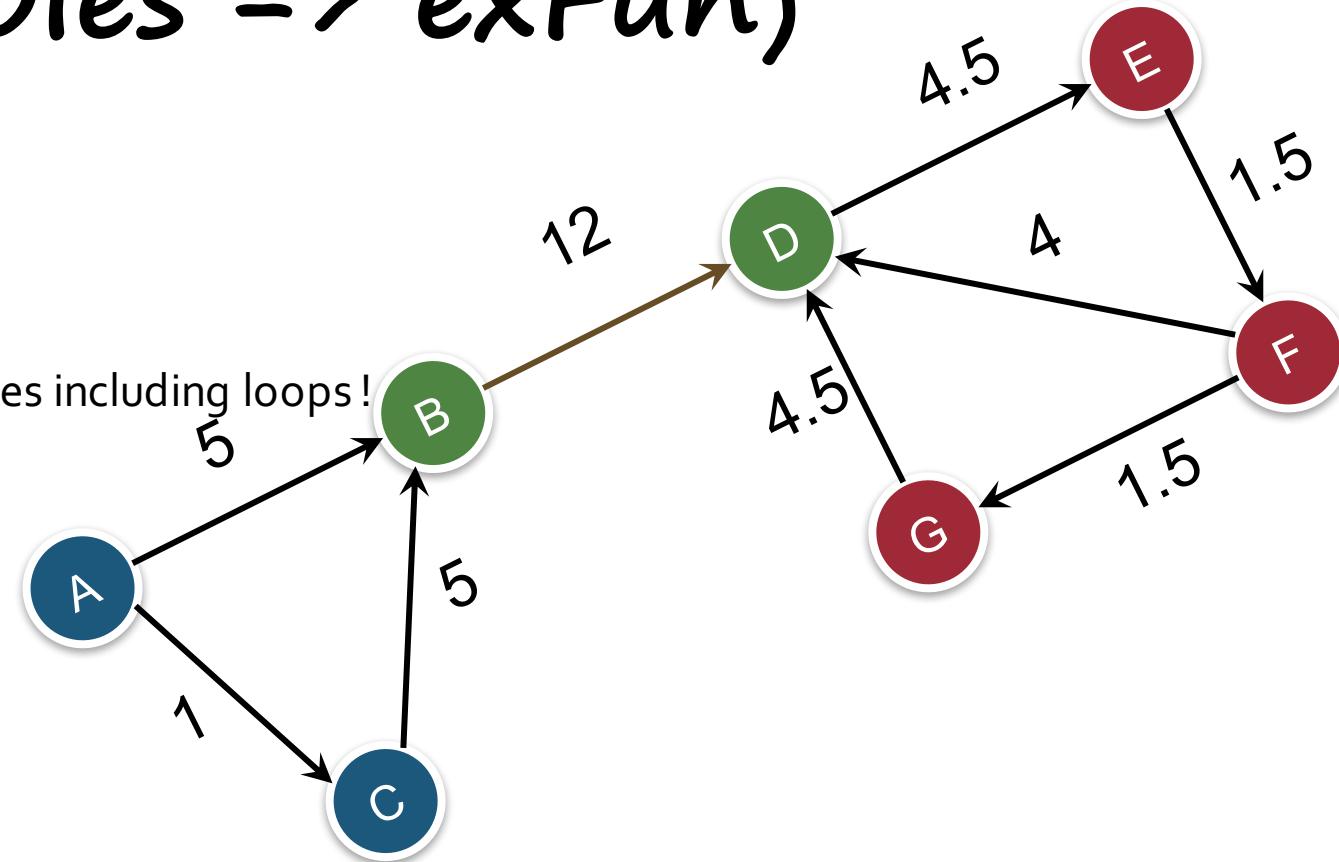
Structural Queries	Indegrees, vertices,...
Attribute Transformers	mapVertices, ...
Structure Transformers	Reverse, subgraph
Join	
Algorithms	Aggregate messages, PageRank

Interesting Objects

- *EdgeTriplet*
- *EdgeContext*

API.inAction(examples => exFun)

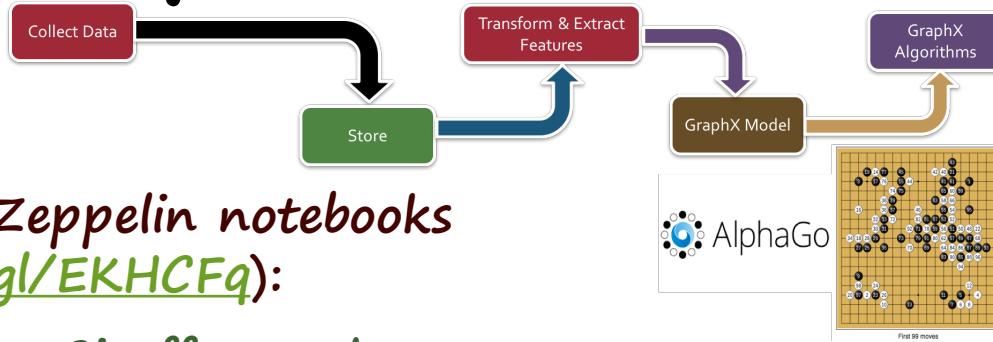
- Computational Model
 - Directed MultiGraph
 - Directed - so in-degree & out-degree
 - MultiGraph – so multiple parallel edges between nodes including loops!
 - Algorithms beware – can cyclic, loops
 - Property Graph
 - vertexID(64bit long) int
 - Need property, cannot ignore it
 - Vertex, Edge Parameterized over object types
 - Vertex[(VertexId, VD)], Edgeg[ED]
 - Attach user-defined objects to edges and vertices (ED/VD)



Good exercises <https://www.sics.se/~amir/files/download/dic/answers6.pdf>

Graphs and Social Networks-Prof. Jeffrey D. Ullman Stanford University
<http://web.stanford.edu/class/cs246/handouts.html>
G-N Algorithm for inbetweeness

API.inAction(examples => exFun)



```

val vertexList = List(
  (1L, Person("Alice", 18)),
  (2L, Person("Bernie", 17)),
  (3L, Person("Cruz", 15)),
  (4L, Person("Donald", 12)),
  (5L, Person("Ed", 15)),
  (6L, Person("Fran", 10)),
  (7L, Person("Genghis", 854)))
  
```

- For our hands-on we will run thru 2 Zeppelin notebooks (<https://goo.gl/qCwZiq> & <https://goo.gl/EKHCFq>):

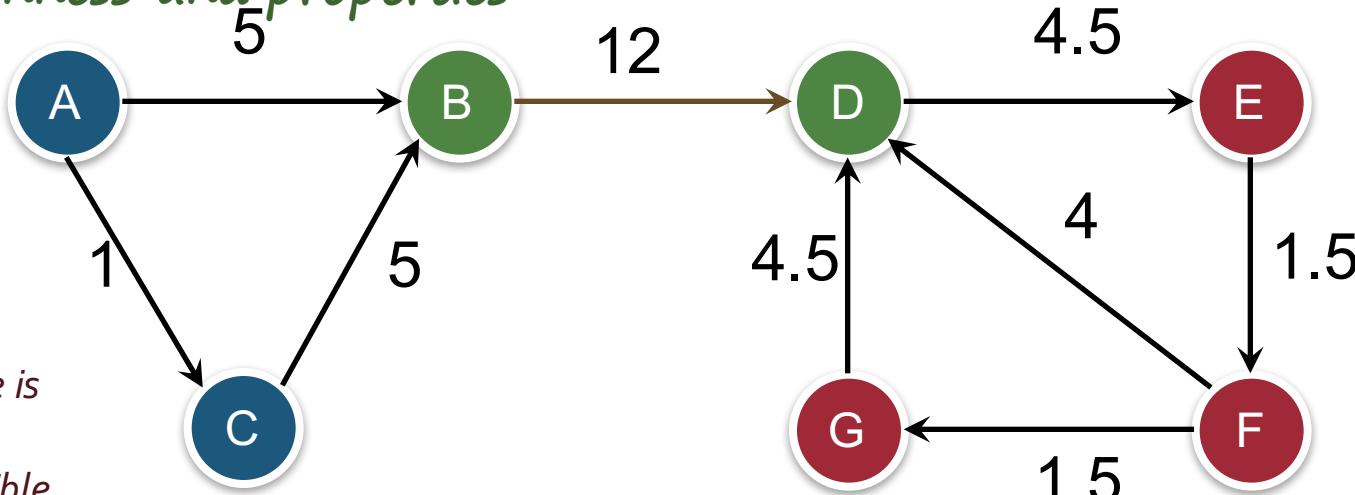
a. First we will work with the following Giraffe graph

- ... carefully chosen with interesting betweenness and properties
- ... for understanding the APIs

b. Second, we will develop a retweet pipeline

- With the AlphaGo twitter topic
- 2 GB/330K tweets, 200K edges,...

Fun facts about inbetweenness : It shows how many paths an edge is part of. High betweenness is the sign of a bottleneck – these edges need HA, probably need alternate paths for re routing, are susceptible to parasite infections and good candidates for a cut!



Graphs and Social Networks-Prof. Jeffrey D. Ullman Stanford University
<http://web.stanford.edu/class/cs246/handouts.html>
 G-N Algorithm for inbetweeness

Good exercises <https://www.sics.se/~amir/files/download/dic/answers6.pdf>

Community-Affiliation-Strengths

- Applied in many ways
- For example in Fraud & Security Applications
 - Triangle detection – for spam servers
- The age of a community is related to the density of triangles
 - New community will have few triangles, then triangles start to form
- Strong affiliation ie Heavy hitter = \sqrt{m} degrees!
 - Heavy hitter triangle !

Algorithms

- Graph-Parallel Computation
 - aggregateMessages() Function
 - Pregel() (<https://issues.apache.org/jira/browse/SPARK-5062>)
- pageRank()
 - Influential papers in a citation network
 - Influencer in retweet
- staticPageRank()
 - Static no of iterations, dynamic tolerance – see the parameters (tol vs. numIter)
- personalizedPageRank()
 - Personalized PageRank is a variation on PageRank that gives a rank relative to a specified “source” vertex in the graph – “People You May Know”
- shortestPaths, SVD++,
 - LabelPropagation (LPA) as described by Raghavan et al in their 2007 paper “Near linear time algorithm to detect community structures in large-scale networks”
 - The Global Clustering Coefficient, is better in that it always returns a number between 0 and 1
 - For comparing connectnedness between different sized communities

```
def aggregateMessages[A](sendMsg: (EdgeContext[VD, ED, A]) => Unit, mergeMsg: (A, A) => A,  
  tripletFields: TripletFields = TripletFields.All)(implicit arg0: ClassTag[A]): VertexRDD[A]
```

Aggregates values from the neighboring edges and vertices of each vertex. The user-supplied sendMsg function is invoked on each edge of the graph, generating 0 or more messages to be sent to either vertex in the edge. The mergeMsg function is then used to combine all messages destined to the same vertex.

A the type of message to be sent to each vertex

sendMsg runs on each edge, sending messages to neighboring vertices using the [EdgeContext](#).

mergeMsg used to combine messages from sendMsg destined to the same vertex. This combiner should be commutative and associative.

tripletFields which fields should be included in the [EdgeContext](#) passed to the sendMsg function. If not all fields are needed, specifying this can improve performance.

▪ Versatile Function useful for implementing PageRank et al

- Can be difficult at first, but easier to comprehend if treated as a combined map-reduce function (it was called **MapReduceTriplets !! With a slightly different signature**)

- aggregateMessage[Msg] (

- map(edgeContext=> mapFun, <- this can be up or down ie sendToDst or Src!)
 - reduce(Msg,Msg) => reduceFun)

attr: ED

The attribute associated with the edge.

dstAttr: VD

The vertex attribute of the edge's destination vertex.

dstId: [VertexId](#)

The vertex id of the edge's destination vertex.

sendToDst(msg: A): Unit

Sends a message to the destination vertex.

sendToSrc(msg: A): Unit

Sends a message to the source vertex.

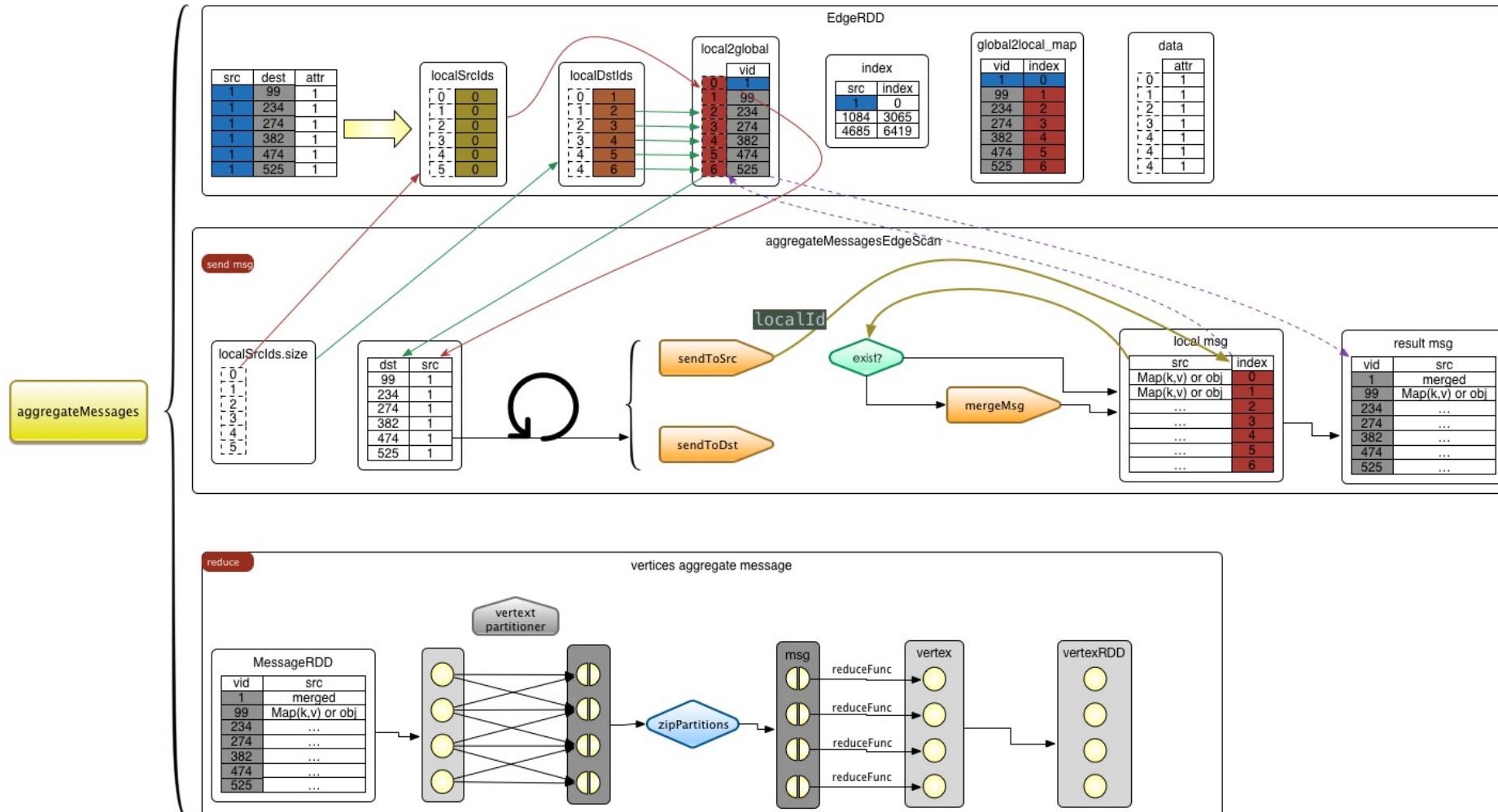
srcAttr: VD

The vertex attribute of the edge's source vertex.

srcId: [VertexId](#)

The vertex id of the edge's source vertex.

If you really want to know what is underneath the aggregateMessages()...



```
val oldestFollower = graph.aggregateMessages[Int]{
  edgeContext => edgeContext.sendToDst(edgeContext.srcAttr.age), //sendMsg
  (x,y) => math.max(x,y) //mergeMsg
}
oldestFollower.collect()
```

→ sendToDst vs sendToSrc

```
oldestFollower: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2056] at RDD at VertexRDD.scala:57
res79: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,854), (5,12), (6,15), (2,18), (3,18), (7,10))
```

```
val oldestFollowee = graph.aggregateMessages[Int]{
  edgeContext => edgeContext.sendToSrc(edgeContext.srcAttr.age), //sendMsg
  (x,y) => math.max(x,y) //mergeMsg
}
oldestFollowee.collect()
```

```
oldestFollowee: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2092] at RDD at VertexRDD.scala:57
res99: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,12), (1,18), (5,15), (6,10), (2,17), (3,15), (7,854))
```

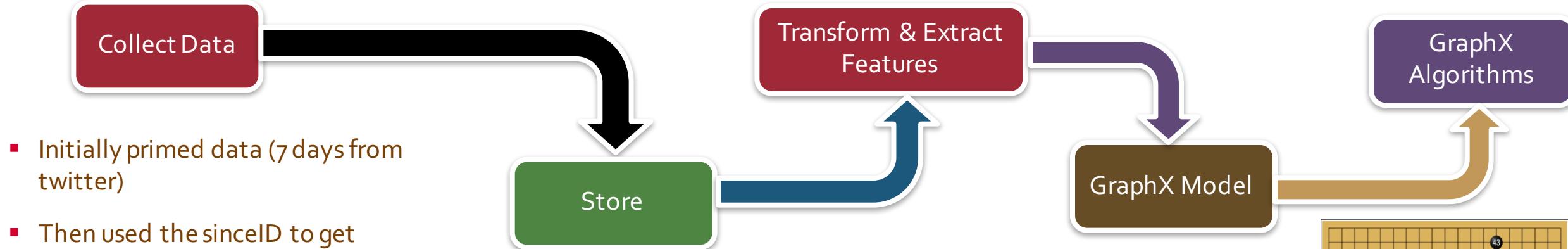
```
val oDegree = graph.aggregateMessages[Int]{
  edgeContext => edgeContext.sendToSrc(1), //sendMsg
  (x,y) => x+y //mergeMsg
}
oDegree.collect()
graph.outDegrees.collect()
```

```
oDegree: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2100] at RDD at VertexRDD.scala:57
res104: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (1,2), (5,1), (6,2), (2,1), (3,1), (7,1))
res105: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (1,2), (5,1), (6,2), (2,1), (3,1), (7,1))
```

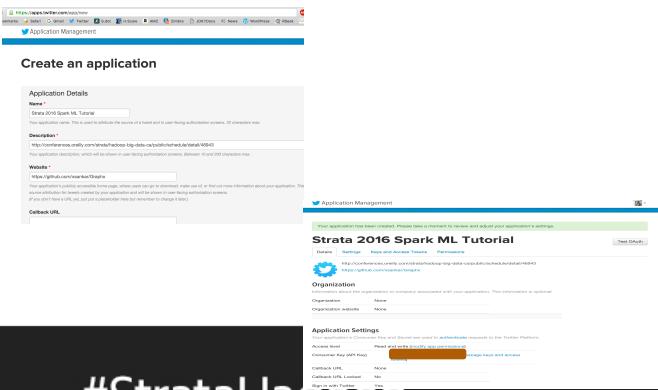
```
val iDegree = graph.aggregateMessages[Int]{
  edgeContext => edgeContext.sendToDst(1), //sendMsg
  (x,y) => x+y //mergeMsg
}
iDegree.collect()
graph.inDegrees.collect()
```

```
iDegree: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[2096] at RDD at VertexRDD.scala:57
res101: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (5,1), (6,1), (2,2), (3,1), (7,1))
res102: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,3), (5,1), (6,1), (2,2), (3,1), (7,1))
```

AlphaGo Tweets Analytics Pipeline

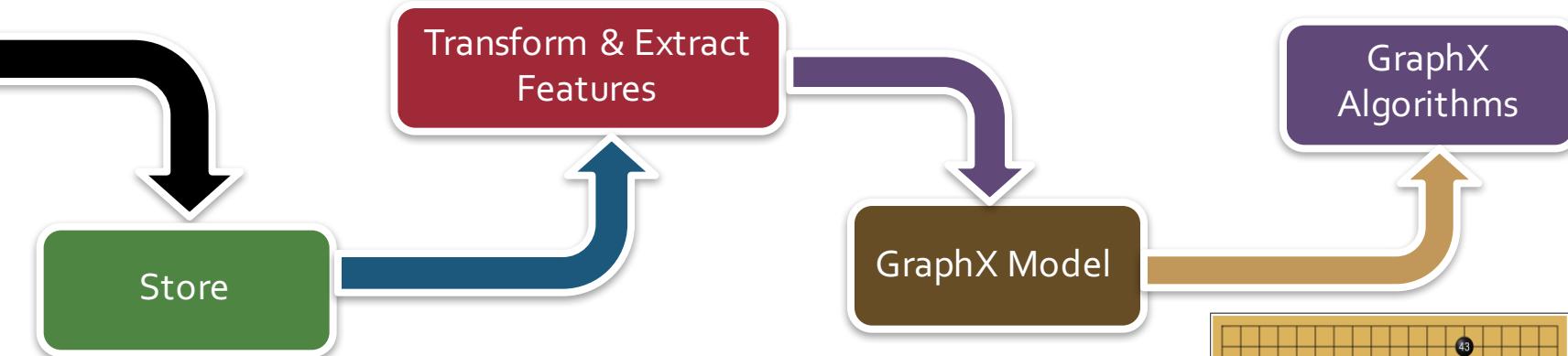


- Initially primed data (7 days from twitter)
- Then used the sinceID to get incremental tweets
- *Use application authentication for higher rate*
- *Used tweepy, wait_on_rate_limit=True, wait_on_rate_limit_notify=True*



#StrataHadoop

- ~330K tweets (see Download program), 2GB
- => MongoDB ~820 MB w/compression)
- Rich data (see MongoDB)
 - Retweet, user ids, hash tags, locations et al
- This exercise only covers the retweet interest network



First 99 moves

```
US-RefConnScreen-201  
2015-03-13T21:51:04.377-0700 dropping: admin.alphago  
2015-03-13T21:51:07.377-0700 [.....] admin.alphago 37.0 MB/1.1 GB (3.3%)  
2015-03-13T21:51:10.377-0700 [.....] admin.alphago 73.0 MB/1.1 GB (6.0%)  
2015-03-13T21:51:13.377-0700 [.....] admin.alphago 109.5 MB/1.1 GB (9.9%)  
2015-03-13T21:51:16.366-0700 [#,.....] admin.alphago 109.5 MB/1.1 GB (9.9%)  
2015-03-13T21:51:19.464-0700 [##,.....] admin.alphago 140.0 MB/1.1 GB (13.2%)  
2015-03-13T21:52:58.369-0700 ##### admin.alphago 882.3 MB/1.1 GB (77.1%)  
2015-03-13T21:53:01.366-0700 ##### admin.alphago 888.5 MB/1.1 GB (80.4%)  
2015-03-13T21:53:04.366-0700 ##### admin.alphago 888.5 MB/1.1 GB (80.4%)  
2015-03-13T21:53:07.366-0700 ##### admin.alphago 888.5 MB/1.1 GB (80.4%)  
2015-03-13T21:53:10.366-0700 ##### admin.alphago 924.5 MB/1.1 GB (83.7%)  
2015-03-13T21:53:13.366-0700 ##### admin.alphago 960.0 MB/1.1 GB (88.6%)  
2015-03-13T21:53:16.369-0700 ##### admin.alphago 1.6 GB/1.1 GB (91.0%)  
2015-03-13T21:53:19.376-0700 ##### admin.alphago 1.1 GB/1.1 GB (99.6%)  
2015-03-13T21:53:22.376-0700 ##### admin.alphago 1.1 GB/1.1 GB (100.0%)  
2015-03-13T21:53:25.371-0700 ##### admin.alphago 1.1 GB/1.1 GB (100.0%)  
2015-03-13T21:53:28.223-0700 ##### admin.alphago 1.1 GB/1.1 GB (100.0%)  
2015-03-13T21:53:28.223-0700 imported 23226 documents  
> db.stats()  
> /usr/local/mongo/bin/mongorestore --db admin --collection alphago --drop --file tweets-20160313.txt
```

1-Extract Tweets

```
%pyspark
import tweepy

apiKey = [REDACTED]
apiSecret = [REDACTED]
accessToken = [REDACTED]
accessTokenSecret = [REDACTED]

auth = tweepy.AppAuthHandler(apiKey, apiSecret) # OAuthHandler(apiKey, apiSecret)
# auth.set_access_token(accessToken, accessTokenSecret)
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

if (not api):
    print ("Can't Authenticate")
    sys.exit(-1)

# http://www.karabbelkar.info/2015/01/how-to-use-twitters-search-rest-api-most-effectively./
```

```
Downloading max 10000000 tweets
Downloaded 100 tweets
Downloaded 200 tweets
Downloaded 300 tweets
Downloaded 400 tweets
Downloaded 500 tweets
Downloaded 600 tweets
Downloaded 700 tweets
Downloaded 800 tweets
Downloaded 900 tweets
Downloaded 1000 tweets
Downloaded 1100 tweets
Downloaded 1200 tweets
Downloaded 1300 tweets
Downloaded 1400 tweets
Downloaded 1500 tweets
Downloaded 1600 tweets
Downloaded 1700 tweets
Downloaded 1800 tweets
Downloaded 1900 tweets
Downloaded 2000 tweets
Downloaded 2100 tweets
Downloaded 2200 tweets
Downloaded 2300 tweets
Downloaded 2400 tweets
Downloaded 2500 tweets
Downloaded 2600 tweets
Downloaded 2700 tweets
Downloaded 2800 tweets
Downloaded 2900 tweets
Downloaded 3000 tweets
Downloaded 3100 tweets
Downloaded 3200 tweets
Downloaded 3300 tweets
Downloaded 3400 tweets
Downloaded 3500 tweets
Downloaded 3600 tweets
Downloaded 3700 tweets
Downloaded 3800 tweets
Downloaded 3900 tweets
Downloaded 4000 tweets
Downloaded 4100 tweets
Downloaded 4200 tweets
Downloaded 4300 tweets
Downloaded 4400 tweets
Downloaded 4500 tweets
Downloaded 4600 tweets
Downloaded 4700 tweets
Downloaded 4800 tweets
Downloaded 4900 tweets
Downloaded 5000 tweets
Downloaded 5100 tweets
No more tweets found
Downloaded 5166 tweets, Saved to tweets.txt
```

```
%pyspark
import sys
import jsonpickle
import os

searchQuery = 'AlphaGo' # this is what we're searching for
maxTweets = 10000000 # Some arbitrary large number
tweetsPerQry = 100 # this is the max the API permits
fName = 'tweets.txt' # We'll store the tweets in a text file.

# If results from a specific ID onwards are reqd, set since_id to that ID.
# else default to no lower limit, go as far back as API allows
sinceId = 712410062820511744 # 711720090954108928 # 710883718903140353 # 710312094227300352 # 709817136437403649# 709550216467185664 # 70921104719872000L# None

# If results only below a specific ID are, set max_id to that ID.
# else default to no upper limit, start from the most recent tweet matching the search query.
max_id = -1L

tweetCount = 0
print("Downloading max {0} tweets".format(maxTweets))
with open(fName, 'w') as f:
    while tweetCount < maxTweets:
        try:
            if (max_id <= 0):
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry)
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry, since_id=sinceId)
            else:
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry, max_id=str(max_id - 1))

            if len(new_tweets) == 0:
                break
```

2-Pipeline screen shots

- Get max tweet id
 - db.alphago.find().sort({id:-1}).limit(1).pretty()
 - "id" : NumberLong("709550216467185664")
 - db.alphago.find().sort({id:+1}).limit(1).pretty()
 - "id" : NumberLong("709211132498714627")
 - /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160316.txt
 - 232296
 - Min : "id" : NumberLong("705845567537221632")
 - Max : "id" : NumberLong("709211104719872000")
 - Count : 253664
 - /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160317.txt
 - +21368
 - Min : "id" : NumberLong("705845567537221632")
 - Max : "id" : NumberLong("709550216467185664")
 - Count : 253664
 - /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160318.txt
 - +10797
 - Min: "id" : NumberLong("705845567537221632")
 - Max: "id" : NumberLong("710883718903140353")
 - Count : 320244
 - /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160320.txt
 - +11511
 - Min: "id" : NumberLong("705845567537221632")
 - Max: "id" : NumberLong("711720090954108928")
 - Count : 331755
 - /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160322.txt
 - +5166
 - Min: "id" : NumberLong("705845567537221632")
 - Max: "id" : NumberLong("712410062820511744")
 - Count : 336921
- ```
USS-Defiant:Strata-2016 ksankar$ /usr/local/mongo/bin/mongoexport --db admin --collection alphago --fieldFile export_fields.txt -o data-01.json
2016-03-13T23:07:30.210-0700 connected to: localhost
2016-03-13T23:07:31.210-0700 [....] admin.alphago 0/232296 (0.0%) ▶ /usr/local/mongo/bin/mongoimport --db admin --collection alphago --file tweets-20160316.txt
2016-03-13T23:07:31.210-0700 [....] admin.alphago 0/232296 (0.0%)
2016-03-13T23:07:31.210-0700 [....] admin.alphago 0/232296 (0.0%)
2016-03-13T23:07:34.221-0700 [....] admin.alphago 0/232296 (0.0%)
2016-03-13T23:07:35.220-0700 [....] admin.alphago 0/232296 (0.0%)
2016-03-13T23:07:36.219-0700 [....] admin.alphago 8000/232296 (3.4%)
2016-03-13T23:07:37.221-0700 [....] admin.alphago 8000/232296 (3.4%)
2016-03-13T23:07:38.220-0700 [....] admin.alphago 8000/232296 (3.4%)
2016-03-13T23:07:39.221-0700 [#....] admin.alphago 16000/232296 (6.9%)
2016-03-13T23:07:40.221-0700 [#....] admin.alphago 16000/232296 (6.9%)
2016-03-13T23:07:41.222-0700 [#....] admin.alphago 16000/232296 (6.9%)
2016-03-13T23:07:42.221-0700 [#....] admin.alphago 16000/232296 (6.9%)
2016-03-13T23:07:43.224-0700 [##....] admin.alphago 24000/232296 (10.3%)
2016-03-13T23:07:44.222-0700 [##....] admin.alphago 24000/232296 (10.3%)
2016-03-13T23:07:45.219-0700 [##....] admin.alphago 24000/232296 (10.3%)
2016-03-13T23:07:46.221-0700 [###....] admin.alphago 32000/232296 (13.8%)
2016-03-13T23:07:47.219-0700 [###....] admin.alphago 32000/232296 (13.8%)
2016-03-13T23:09:04.219-0700 [#####....] admin.alphago 184000/232296 (79.2%)
2016-03-13T23:09:05.221-0700 [#####....] admin.alphago 192000/232296 (82.7%)
2016-03-13T23:09:06.222-0700 [#####....] admin.alphago 192000/232296 (82.7%)
2016-03-13T23:09:08.220-0700 [#####....] admin.alphago 192000/232296 (82.7%)
2016-03-13T23:09:08.220-0700 [#####....] admin.alphago 192000/232296 (82.7%)
2016-03-13T23:09:09.222-0700 [#####....] admin.alphago 200000/232296 (86.1%)
2016-03-13T23:09:10.224-0700 [#####....] admin.alphago 200000/232296 (86.1%)
2016-03-13T23:09:11.222-0700 [#####....] admin.alphago 200000/232296 (86.1%)
2016-03-13T23:09:12.221-0700 [#####....] admin.alphago 208000/232296 (89.5%)
2016-03-13T23:09:13.223-0700 [#####....] admin.alphago 208000/232296 (89.5%)
2016-03-13T23:09:14.221-0700 [#####....] admin.alphago 208000/232296 (89.5%)
2016-03-13T23:09:15.220-0700 [#####....] admin.alphago 208000/232296 (89.5%)
2016-03-13T23:09:16.219-0700 [#####....] admin.alphago 208000/232296 (89.5%)
2016-03-13T23:09:17.221-0700 [#####....] admin.alphago 216000/232296 (93.0%)
2016-03-13T23:09:18.219-0700 [#####....] admin.alphago 216000/232296 (93.0%)
2016-03-13T23:09:19.221-0700 [#####....] admin.alphago 216000/232296 (93.0%)
2016-03-13T23:09:19.221-0700 [#####....] admin.alphago 224000/232296 (96.4%)
2016-03-13T23:09:21.223-0700 [#####....] admin.alphago 224000/232296 (96.4%)
2016-03-13T23:09:22.221-0700 [#####....] admin.alphago 224000/232296 (96.4%)
2016-03-13T23:09:23.221-0700 [#####....] admin.alphago 224000/232296 (96.4%)
2016-03-13T23:09:23.446-0700 [#####....] admin.alphago 232296/232296 (100.0%)
2016-03-13T23:09:23.446-0700 [#####....] admin.alphago 232296/232296 (100.0%)
exported 232296 records

```

# 3-Twitter gives lots of fields in Mongo

```
|> db.alphago.find().sort({_id:-1}).limit(1).pretty()
{
 "_id" : ObjectId("56f1cc2a6e19ba0bcd5f9ddc"),
 "contributors" : null,
 "truncated" : false,
 "text" : "RT @cdixon: \"The human mind is still 50,000 times more energy efficient than machine intelligence.\" https://t.co/bChxqspT3M https://t
.co/pp...",
 "is_quote_status" : false,
 "in_reply_to_status_id" : null,
 "in_reply_to_user_id" : null,
 "id" : NumberLong("712410062820511744"),
 "favorite_count" : 0,
 "source" : "Twitte
 "retweeted" : false,
 "coordinates" : null,
 "entities" : {
 "symbols" : [],
 "media" : [
 {
 "source_user_id" : 2529971,
 "source_status_id_str" : "711298715646955520",
 "expanded_url" : "http://twitter.com/cdixon/status/71129871
 "display_url" : "pic.twitter.com/pp5ZtNQMOU",
 "source_status_id" : NumberLong("711298715646955520"),
 "media_url_https" : "https://pbs.twimg.com/media/Cd8KaKEVIAw6mg.jpg"
 "source_user_id_str" : "2529971",
 "url" : "https://t.co/pp5ZtNQMOU",
 "id_str" : "711298710643220480",
 "sizes" : [
 {
 "large" : {
 "h" : 669,
 "w" : 1024,
 "resize" : "fit"
 },
 "small" : {
 "h" : 222,
 "w" : 340,
 "resize" : "fit"
 }
 }
],
 "media" : [
 {
 "resizes" : {
 "large" : {
 "h" : 669,
 "w" : 1024,
 "resize" : "fit"
 },
 "medium" : {
 "h" : 392,
 "w" : 600,
 "resize" : "fit"
 },
 "thumb" : {
 "h" : 150,
 "w" : 150,
 "resize" : "crop"
 }
 },
 "indices" : [
 124,
 140
],
 "type" : "photo",
 "id" : NumberLong("711298710643220480"),
 "media_url" : "http://pbs.twimg.com/media/Cd8KaKEVIAw6mg.jpg"
 }
],
 "hashtags" : [],
 "user_mentions" : [
 {
 "indices" : [
 3,
 10
],
 "screen_name" : "cdixon",
 "id" : 2529971,
 "name" : "Chris Dixon",
 "id_str" : "2529971"
 }
],
 "urls" : [
 {
 "indices" : [
 100,
 123
],
 "url" : "https://t.co/bChxqspT3M",
 "expanded_url" : "http://jacquesmattheij.com/another-way-of-lc
 }
]
 }
],
 "favorited" : false,
 "user" : {
 "follow_request_sent" : null,
 "has_extended_profile" : false,
 "profile_use_background_image" : true,
 "id" : 2529971,
 "verified" : true,
 "profile_text_color" : "333333",
 "profile_image_url_https" : "https://pbs.twimg.com/profile_images/68346924104658944/80a5XAs0_normal.png",
 "profile_sidebar_fill_color" : "DDEEF6",
 "entities" : {
 "url" : {
 "urls" : [
 {
 "indices" : [
 0,
 22
],
 "url" : "http://t.co/VviCcFN0iV",
 "expanded_url" : "http://cdixon.org/aboutme/",
 "display_url" : "cdixon.org/aboutme/"
 }
]
 },
 "description" : {
 "urls" : []
 }
 },
 "followers_count" : 218312,
 "protected" : false,
 "location" : "CA & NYC",
 "default_profile_image" : false,
 "id_str" : "2529971",
 "lang" : "en",
 "utc_offset" : -14400,
 "statuses_count" : 9225,
 "description" : "programming, philosophy, history, internet, startups, investing",
 "friends_count" : 3376,
 "profile_background_image_url_https" : "https://pbs.twimg.com/profile_background_images/543964120664403968/0IKm7siW.png",
 "profile_link_color" : "89C9FA",
 "profile_sidebar_fill_color" : "DDEEF6"
 }
 }
}
```

# 4-Extract Retweet Fields

&

## 5-Store in CSV

```
%pyspark
import pymongo
from pymongo import MongoClient

print " pymongo : %s" % (pymongo.version)
client=MongoClient('localhost', 27017)
db = client["admin"] #db name
alphago=db["alphago"]

cursor = alphago.find({},{"id":1,"text":1,"user.id":1,"user.followers_count":1,"user.friends_count":1,"user.name":1,"user.time_zone":1,"user.location":1,
"retweet_count":1,"retweeted_status.user.id":1,"retweeted_status.user.followers_count":1,"retweeted_status.user.friends_count":1,"retweeted_status.user.name":1,
"retweeted_status.user.time_zone":1,"retweeted_status.user.location":1})
print cursor.count()
```

```
%pyspark
write csv file
count = 0
#with open('reTweetNetwork-small.psv', 'w') as f:
with open('reTweetNetwork-large.psv', 'w') as f:
 for doc in cursor:
 tweetId = doc["id"]
 tweetCount = doc["retweet_count"]
 tweetText = "NA" #doc["text"].encode('ascii', 'ignore').replace('\n', ' ').replace('\r', '')
 srcUserId = doc["user"]["id"]
 srcUserName = doc["user"]["name"].encode('ascii', 'ignore')
 if (srcUserName is not None):
 srcUserName = srcUserName.rstrip().lstrip()
 if (srcUserName is None):
 srcUserName="NA"
 srcUserLocation = doc["user"]["location"].encode('ascii', 'ignore')
 if (srcUserLocation is not None):
 srcUserLocation = srcUserLocation.rstrip().lstrip()
 if (srcUserLocation is None):
 srcUserLocation="NA"
 srcUserTz = doc["user"]["time_zone"]
 if (srcUserTz is not None):
 srcUserTz = srcUserTz.rstrip().lstrip()
 if (srcUserTz is None):
 srcUserTz="NA"
 srcUserFr = doc["user"]["friends_count"]
 if (not (type(srcUserFr) is int)):
 srcUserFr = 0
 srcUserFol = doc["user"]["followers_count"]
 if (not (type(srcUserFol) is int)):
 srcUserFol = 0
 if ("retweeted_status" in doc):
 destUserId = doc["retweeted_status"]["user"]["id"]
 destUserName = doc["retweeted_status"]["user"]["name"].encode('ascii', 'ignore')
 if (destUserName is not None):
```

```

 destUserTz = doc["retweeted_status"]["user"]["time_zone"]
 if (destUserTz is not None):
 destUserTz = destUserTz.rstrip().lstrip()
 if (destUserTz is None):
 destUserTz = "NA"
 destUserFr = doc["retweeted_status"]["user"]["friends_count"]
 destUserFol = doc["retweeted_status"]["user"]["followers_count"]
 else:
 destUserId = "NA"
 destUserName = "NA"
 destUserLocation = "NA"
 destUserTz = "NA"
 destUserFr = "NA"
 destUserFol = "NA"
 if ("retweeted_status" in doc): # write only retweets, as we want to create the retweet network
 f.write('{0}|{1}|{2}|{3}|{4}|{5}|{6}|{7}|{8}|{9}|{10}|{11}|{12}|{13}|{14}\n'.format(tweetId, tweetCount, tweetText, srcUserId, srcUserName,
 srcUserLocation, srcUserTz, srcUserFr, srcUserFol, destUserId, destUserName, destUserTz, destUserFr, destUserFol))
 # count += 1
 #if (count > 10000):
 # break
 print "*** Done ***"
```

```

val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val df = sqlContext.read
 .format("com.databricks.spark.csv")
 .option("header", "false") // Use first line of all files as header
 .option("inferSchema", "true") // Automatically infer data types
 .option("delimiter", ",")
 .load("reTweetNetwork-small.psv")

df.show(5)
df.count()

sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@29b11
df: org.apache.spark.sql.DataFrame = [C0: bigint, C1: int, C2: string, C4: string, C5: string, C6: string, C7: int, C8: int, C9: bigint, C10: string, C11: string, C12: string, C13: int, C14: int]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| C0 | C1 | C2 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1709210987015114752	2	NAI	589277295	Unsigned Heroes!	Seattle	WA	Arizona	91912703	12798452	Erick Schonfeld!	New York	Central Time
1709210994875039744	25	NAI	271007340	Johnny Apuan!	None	1252	99	1344951	WIRED	San Francisco/New...	Pacific Time	
1709210924771467264	39	NAI	166001040	Nedpool Canterbury New Ze...	Wellington	1033	572	8917142	Dan Kaminsky	Chief Scientist, ...	Pacific Time	
1709210923928559616	25	NAI	3244208187	Deen over Duniyal	Mars	None	93	223	1344951	WIRED	San Francisco/New...	Pacific Time
1709210919729963008	25	NAI	274975425	Farzan Sabet!	Eastern Time	CUS ...	373	862	1344951	WIRED	San Francisco/New...	Pacific Time
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```

// Create vertices & edges
case class User(name:String, location:String, tz : String, fr:Int, fol:Int)
case class Tweet(id:String, count:Int)

val graphData = df.rdd
graphData.take(2)

val vert1 = graphData.map(row => (row(3).toString.toLong,User(row(4).toString, row(5).toString, row(6).toString, row(7).toString.toInt, row(8).toString.toInt)))
vert1.count()
vert1.take(3)

val vert2 = graphData.map(row => (row(9).toString.toLong,User(row(10).toString, row(11).toString, row(12).toString, row(13).toString.toInt, row(14).toString.toInt)))
vert2.count()
vert2.take(3)

val vertX = vert1.++(vert2)
vertX.count()

val edgX = graphData.map(row => (Edge(row(3).toString.toLong, row(9).toString.toLong,(row(0),row(1)))))
edgX.take(3)

val rtGraph = Graph(vertX,edgX)

```

# 6-Read as dataframe

# 7-Create Vertices, Edges & Objects

# 8-Finally the graph

# & run algorithms

```

val ranks = rtGraph.pageRank(0.1).vertices
ranks.take(2)
val topUsers = ranks.sortBy(_._2, false).take(3).foreach(println)
val topUsersWNames = ranks.join(rtGraph.vertices).sortBy(_._2._1, false).take(3).foreach(println)

ranks: org.apache.spark.graphx.VertexRDD[Double] = VertexRDDImpl[627] at RDD at VertexRDD.scala:57
res92: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((144366820,0.15), (9734132,0.15))
(11821362,322.9447926136434)
(1482581556,84.10384874154467)
(14497118,43.956874999999826)
topUsers: Unit = ()
(11821362,(322.9447926136434,User(Statsman Bruno,Stockholm,Stockholm,53,79094)))
(1482581556,(84.10384874154467,User(Demis Hassabis,,None,11,22637)))
(14497118,(43.956874999999826,User(Ken Kawamoto,Kokubunji, Tokyo,Tokyo,549,5675)))
topUsersWNames: Unit = ()
Took 3 seconds

```

How many retweets ?

```

Took 0 seconds
val iDeg = rtGraph.inDegrees
val oDeg = rtGraph.outDegrees
iDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[648] at RDD at VertexRDD.scala:57
oDeg: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[475] at RDD at VertexRDD.scala:57
Took 1 seconds

iDeg.take(3)
iDeg.sortBy(_._2, false).take(3).foreach(println)
iDeg
res36: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((630222308,1), (149925820,11), (169630984,2))
res37: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((11821362,2574), (1482581556,860), (14497118,360))
Took 1 seconds (outdated)

```

# Wikipedia Page Rank Analysis