Part 8: Analyzing Graphs, Redux (2/2)

November 21, 2019

Ali Abedi

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details
Theme for Today:

How things work in the real world
(forget everything you’ve been told...)
(these are the mostly true events of Jimmy Lin’s Twitter tenure)
From the Ivory Tower...
... to building sh*t that works
What exactly might a data scientist do at Twitter?
They might have worked on...
– analytics infrastructure to support data science
– data products to surface relevant content to users


They might have worked on...
- analytics infrastructure to support data science
- data products to surface relevant content to users
circa ~2010
~150 people total
~60 Hadoop nodes
~6 people use analytics stack daily

circa ~2012
~1400 people total
10s of Ks of Hadoop nodes, multiple DCs
10s of PBs total Hadoop DW capacity
~100 TB ingest daily
dozens of teams use Hadoop daily
10s of Ks of Hadoop jobs daily
WTF

(who to follow)

Who to follow · refresh · view all

- freshbooks FreshBooks · Follow
  - Promoted · Followed by @zappos and others.
- alanwarms Alan Warms · Follow
  - Followed by @fredwilson and others.
- Mozzie21 Moises Henriques · Follow
  - can eat

Similar to @ryanhall3 · view all

- RunnerSpace_com RunnerSpace.com · Follow
  - RunnerSpace.com has the latest in news and media...
- chrislieto chris lieto · Follow
  - Chris Lieto is a top ranked World Class Triathlete, ...
- runningtimes runningtimes · Follow
#numbers
(Second half of 2012)

~175 million active users
~20 billion edges
42% edges bidirectional
Avg shortest path length: 4.05
40% as many unfollows as follows daily
WTF responsible for ~1/8 of the edges

Myers, Sharma, Gupta, Lin. Information Network or Social Network? The Structure of the Twitter Follow Graph. WWW 2014.
Graphs are core to Twitter

Graph-based recommendation systems
Why? Increase engagement!
The Journey

From the static follower graph for account recommendations...
... to the real-time interaction graph for content recommendations

In Four Acts...
In the beginning... the void

Act I
WTF and Cassovary
(circa 2010)
In the beginning... the void
Goal: build a recommendation service quickly

Act I
WTF and Cassovary
(circa 2010)
flockDB
(graph database)

Simple graph operations
Set intersection operations

Not appropriate for graph algorithms!
Okay, let’s use MapReduce!
But MapReduce sucks for graphs!
MapReduce sucks for graph algorithms...
Let’s build our own system!

Key design decision:
Keep entire graph in memory... on a single machine!
Nuts!

Why?
Because we can!
Graph partitioning is hard... so don’t do it
Simple architecture

Right choice at the time!

Source: Wikipedia (Pistachio)
Suppose: $10 \times 10^9$ edges
(src, dest) pairs: $\sim 80$ GB

$18 \times 8$ GB DIMMS = 144 GB
$18 \times 16$ GB DIMMS = 288 GB
$12 \times 16$ GB DIMMS = 192 GB
$12 \times 32$ GB DIMMS = 384 GB
Cassovary

In-memory graph engine

- Implemented in Scala
- Compact in-memory representations
- But no compression
- Avoid JVM object overhead!
- Open-source

Source: Wikipedia (Cassowary)
PageRank

“Semi-streaming” algorithm
Keep vertex state in memory, stream over edges
Each pass = one PageRank iteration
Bottlenecked by memory bandwidth

Convergence?
Don’t run from scratch… use previous values
A few passes are sufficient
“Circle of Trust”

Ordered set of important neighbors for a user
Result of egocentric random walk: Personalized PageRank!
Computed online based on various input parameters

One of the features used in search
SALSA for Recommendations

"hubs"

CoT of $u$

users LHS follow

"authorities"

hubs scores: similarity scores to $u$

authority scores: recommendation scores for $u$
What about new users?

Cold start problem: they need recommendations the most!
Spring 2010: no WTF
   seriously, WTF?

Summer 2010: WTF launched
Another “interesting” design choice:
We migrated from Cassovary back to Hadoop!
Whaaaaaaaa?
Cassovary was a stopgap!

Hadoop provides:
Richer graph structure
Simplified production infrastructure
Scaling and fault-tolerance “for free”

Right choice at the time!
Wait, didn’t you say MapReduce sucks?

What exactly is the issue?
Random walks on egocentric 2-hop neighborhood
Naïve approach: self-joins to materialize, then run algorithm

The shuffle is what kills you!
Graph algorithms in MapReduce

Tackle the shuffling problem!

Key insights:
Batch and “stich together” partial random walks*
Clever sampling to avoid full materialization

* Sarma et al. Estimating PageRank on Graph Streams. PODS 2008
Throw in ML while we’re at it...

Follow graph  Retweet graph  Favorite graph  ...

Candidate Generation  Trained Model

Candidates  Classification  Final Results

Act III

MagicRecs

(circa 2013)
Isn’t the point of Twitter real-time? 
So why is WTF still dominated by batch processing?
Observation: fresh recommendations get better engagement

Logical conclusion: generate recommendations in real time!

From batch to real-time recommendations:
Recommended based on recent activity
“Trending in your network”

Inverts the WTF problem:
For this user, what recommendations to generate?
Given this new edge, which user to make recommendations to?
Why does this work?
A follows B’s because they’re interesting
B’s following C’s because “something’s happening”
(generalizes to any activity)

Scale of the Problem
O(10^8) vertices, O(10^{10}) edges
Designed for O(10^4) events per second

Naïve solutions:
Poll each vertex periodically
Materialize everyone’s two-hop neighborhood, intersect

Production solution:
Idea #1: Convert problem into adjacency list intersection
Idea #2: Partition graph to eliminate non-local intersections

Single Node Solution

Who we’re recommending

“influencers”

Who we’re making the recommendations to

D “dynamic” structure:
stores inverted adjacency lists
query C, return all B’s that link to it

S “static” structure:
stores inverted adjacency lists
query B, return all A’s that link to it
Algorithm

D “dynamic” structure:
stores inverted adjacency lists
query C, return all B’s that link to it

1. Receive $B_3$ to $C_2$
2. Query D for $C_2$, get $B_1$, $B_2$, $B_3$
3. For each $B_1$, $B_2$, $B_3$, query S
4. Intersect lists to compute A’s

S “static” structure:
stores inverted adjacency lists
query B, return all A’s that link to it

Who we’re recommending

Idea #1: Convert problem into adjacency list intersection
Who we’re recommending “influencers”

Who we’re making the recommendations to

Distributed Solution

1. Fan out new edge to every node
2. Run algorithm on each partition
3. Gather results from each partition

Idea #2: Partition graph to eliminate non-local intersections
Production Status
Launched September 2013

Usage Statistics (Circa 2014)
Push recommendations to Twitter mobile users
Billions of raw candidates, millions of push notifications daily

Performance
End-to-end latency (from edge creation to delivery):
median 7s, p99 15s
Fully bought into the potential of real-time... but needed something more general

Focused specifically on the interaction graph
Takeaway lesson #01:
Make things as simple as possible, but not simpler.

With lots of data, algorithms don’t really matter that much. Why a complex architecture when a simple one suffices?
Takeaway lesson #10: Constraints aren’t always technical.
Takeaway lesson #11:
Visiting and revisiting design decisions

Source: https://www.flickr.com/photos/exmachina/8186754683/
Questions?

“In theory, there is no difference between theory and practice. But, in practice, there is.”

- Jan L.A. van de Snepscheut