Extracting Information from Social Networks

Reminder: Social networks

- Catch-all term for
 - social networking sites
 Facebook
 - microblogging sites
 - Twitter
 - blog sites (for some purposes)

Ways we can use social networks to find information

- ✓ Extract meta-information for "regular" Web search
 - site information
 - site properties
- · Extract information to use directly
 - search content of social site
 - aggregate information from site content
 - $\mbox{ information from structure of social network}$

Searching social network content

2

- How does searching a social network site differ from searching the Web with a SE?
- Does this affect
 - indexing?
 - query evaluation?
- social site Facebook
- · microblog site Twitter

Searching Facebook

- search for objects (e.g. people) as well as information
- focused searches
 - people
 - friends
 - photos
- · link structure central
 - find friends who ...
- updates important
- other?

Searching Twitter vs Web: User behavior

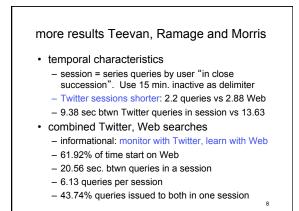
Study by Teevan, Ramage and Morris pub. 2011

Experimental setup

- data from browser logs from Bing Toolbar
- harvest queries issued to search engines
 - "general purpose": Bing, Google, Yahoo
 - "vertical search engines": Twitter
 - associate with user IDs and timestamps
- Sampled 126,316 queries to Twitter
 subset of 33,405 users
- 2.5 million queries by same subset users from Bing,
- Google, Yahoo

Teevan et al results

- unsurprising:
 - top 10 Web searches navigational
 - top 10 Twitter queries mixed celebrities, movies, games, memes (eg "#theresway2many"): popular items
- · more surprising:
 - 23.19% Twitter queries issued only once, vs 49.73% Web
 - 55.76% Twitter queries issued more than once by same user, vs 34.71% Web



Twitter characteristics that may change search approach?

- history more important Twitter findings
- recency more imporant trending
- · popularity more important?
- · labels available hashtags
- · other?

Searching Social Networks: system demands

- Twitter Earlybird 2012
- Facebook Unicorn 2013

Earlybird: Real-Time Search at Twitter by many Twitter researchers (2012)

- Designed for properties of tweets
 - Handle high rate of queries
 - Handle large number updates in real time
 "Flash crowds"
 - Update info, eg number of retweets
 - Large number concurrent reads and writes
 - Time stamp dominant ranking signal

11

Elements

10

12

- Distributed server architecture
 Tweets hash partitioned across servers
- New concurrency management
- Customized query processing
- Customized inverted index

Query processing

- ≻ Full Boolean query language
- Results returned most recent first
- Personalized signals in relevance algorithm (not described)
 - User's local social graph
 - "actual query algorithm isn't particularly interesting"
 - "reuse existing Lucene query eval code"

Inverted Index

- · Organized in segments
 - Each server has small number segments (12)
 - Each segment has small number tweets, $\leq 2^{23}$
 - Only one segment active
 - can modify
 - In-active segments read-only
 Optimize for compression and query eval

Dictionary

- Hash table
- No binary search
- Term => term ID
 - Monotonically increasing in order seen
- Parallel arrays for data
 - Number of postings in postings list for term
 - Pointer to tail of the postings list
 - Each array indexed by term ID

15

13

Active segment index

- Posting is 32-bit integer
 24 bits doc ID; 8 bits term position
 - each occurrence in tweet is new posting
- Postings list: pre-allocated integer array

 Dynamic allocation
- Traversing newest first = iterate bkwds
- Can traverse bkwds from any point while concurrently adding new postings
- Can binary search for doc ID

 Eliminate need skip pointers

Dynamic space allocation

- · Uses 4 dynamic arrays called pools
 - A pools holds "slices" of a certain sizeA slice is part of a postings list
 - Slice sizes 21, 24, 27, 211
- · A posting list starts in a slice of the smallest pool
- · When fills slice in a pool, continue list in larger pool
- · Can use many slices in largest pool
- Slices linked together with pointers: large to small
- Tail of postings list in largest pool occupied

17

In-active segments

- Replaces an active segment when done
- · Fixed-size integer array
- · Arranged reverse chronologically
- · Compressed
 - Short postings list: as before
 - Long postings list:
 - uses gaps
 - block-based compression

14

16

Earlybird performance

- · Compare prior MySQL-based - 1000 tweets per second indexing
 - 12,000 queries per second
- · Earlybird memory
 - Full active index segment (16M tweets) 6.7 GB
 - Full in-active index segment ~ 55% above
- Queries per second - 5000 for fully-loaded server (114M tweets)
- · Tweets per second - 7000 in "stress test"- heavy query load

Unicorn: A System for Searching the Social Graph by many Facebook researchers (2013)

- · primary backend for Facebook Graph Search
- "designed to search trillions of edges between tens of billions of users and entities and entities on thousands of commodity servers"
- thousands of edge types used - including obvious "friend" "like"
- graph sparse:
 - typical node < 1000 edges</p>
 - average user has ~130 friends

Unicorn: graph querying · query language on edge relationships "find female friends of user 6" becomes guery (and friend:6 gender:1) intersection of sets supports queries on paths - rounds of basic query evaluation "find pages liked by friends of user 7 who like Emacs (object 42)" becomes (and friend:7 likers:42) giving {resultID1, ..., resultIDk}

followed by

(or likes:resultID1 ... likes:resultIDk) - does through APPLY operator

(apply likes: (and friend:7 likers:42))

Unicorn APPLY operator

- · applies "or" to results of inner query (apply likes: (and friend:7 likers:42)) can nest APPLY arbitrarily deep
- friends of friends of friends of friends of user 21 (apply friend: (apply friend: (friend 21)))
- · limit on number results of inner query
- solution: drop some results
 - issue: performance
 - cut-off ~100,000 terms applied to outer query

22

20

Unicorn: index struture

- · index represents adjacency list
- index term <edge-type>:<id>
 - friend:5 selects list of friends of userID 5
- · form of adj. list entry:
 - ((sortkey, DocID), other info)
 - nodes on adjacency list sorted first by sortkey, then by nodeID

23

19

21

Unicorn performance query "people who like computer science" > 6 million results - ask for 100 returned run 100 times · average performance - latency 11 ms - aggregate CPU across 37 index servers 31.22 ms

query "friends of likers of computer science"

- for APPLY with trunction limit 10⁵, latency almost 2 sec.
- for APPLY with trunction limit 10³, latency about 100ms