Unicorn: A System for Searching the Social Graph
What is it?

- Primary backed system for Facebook Graph Search
- Billions of nodes
- Trillions of edges
- Sparse
  - Average user has 130 friends

<table>
<thead>
<tr>
<th>Edge-Type</th>
<th># out</th>
<th>in-id-type</th>
<th>out-id-type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>friend</td>
<td>hundreds</td>
<td>USER</td>
<td>USER</td>
<td>Two users are friends (symmetric)</td>
</tr>
<tr>
<td>likes</td>
<td>a few</td>
<td>USER</td>
<td>PAGE</td>
<td>pages (movies, businesses, cities, etc.) liked by a user</td>
</tr>
<tr>
<td>likers</td>
<td>10–10M</td>
<td>PAGE</td>
<td>USER</td>
<td>users who have liked a page</td>
</tr>
<tr>
<td>live-in</td>
<td>1000–10M</td>
<td>PAGE</td>
<td>USER</td>
<td>users who live in a city</td>
</tr>
<tr>
<td>page-in</td>
<td>thousands</td>
<td>PAGE</td>
<td>PAGE</td>
<td>pages for businesses that are based in a city</td>
</tr>
<tr>
<td>tagged</td>
<td>hundreds</td>
<td>USER</td>
<td>PHOTO</td>
<td>photos in which a user is tagged</td>
</tr>
<tr>
<td>tagged-in</td>
<td>a few</td>
<td>PHOTO</td>
<td>USER</td>
<td>users tagged in a photo (see Section 7.2)</td>
</tr>
<tr>
<td>attended</td>
<td>a few</td>
<td>USER</td>
<td>PAGE</td>
<td>schools and universities a user attended</td>
</tr>
</tbody>
</table>
Data Model

- Represented as an adjacency list
- Adjacency lists are referenced by the form
  - `<edge-type>:<id>`
  - Sort-key: most globally important id’s
  - Id: Unique id
  - HitData: Array of bytes that store application-specific data (e.g. graduation year and major)
API and Query Language

- **AND**
  - (and friend:5 gender:1)

- **OR**
  - (or friend:5 friend:6)

- **Difference**
  - (difference (and friend:5 gender:1) (friend:6))
Architecture

- Top-aggregator determines which vertical(s) each query needs to be sent to.
- Rack-aggregator does a retrieval query to each index.
- Each Index Server responds with a result.
- Edges of the same result-type are placed in the same vertical (e.g. friend and likers edge is in the same vertical since they both yield user id’s).
Typeahead

- Dynamically search for entities (users/groups etc)
- Returns a ranked list of possible matches for the query
- No provision for social relevance (and mel* friend:3)
  - WeakAnd and StrongOr

```
(weak-and
  (term friend:3
    :optional-hits 2)
  (term melanie)
  (term mars*))
```

Ensures we do not miss good results who are not friends of 3

```
(strong-or friend:5
  (and friend:5 live-in:100
    :optional-weight 0.2)
  (and friend:5 live-in:101
    :optional-weight 0.1))
```

The optional weights indicates how many percentage of the result must live that city.
Graph Search

- We want to reach nodes more than one edge away
- APPLY
  - Allows clients to use the results of previous executions as seeds for future executions
  - Do not need to denormalize data

<table>
<thead>
<tr>
<th></th>
<th>friend: only</th>
<th>F-o-F inline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Est. #Hits/User</td>
<td>130</td>
<td>48k</td>
</tr>
<tr>
<td>Est. Total Bytes</td>
<td>484GB</td>
<td>178TB</td>
</tr>
<tr>
<td>Avg. Latency</td>
<td>20ms</td>
<td>7ms</td>
</tr>
<tr>
<td>Variance</td>
<td>medium</td>
<td>low</td>
</tr>
</tbody>
</table>

- Friends-of-Friends
  - (apply friend: friend:5)
Privacy

- Certain edges cannot be shown to all users, but rather only users who are friends with the particular person
- Typically the frontend makes a proper privacy check
- Keep the privacy separate from Unicorn: DRY (“Don’t Repeat Yourself”)
  - Because Facebook already have complex privacy logic in the frontend
Conclusion

• Powerful
• Flexible
• Scalable