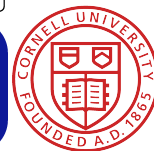


# Search in Social Networks with Access Control





Truls A. Bjørklund, Michaela Götz, Johannes Gehrke  
Norwegian University of Science and Technology, Cornell University










# Content Search in Social Networks



**News Feed**Preferences

 **Rhea Drysdale wrote on Patrick Sexton's wall.**  
 Aw, thank you. :)  
I'm just glad I didn't kill anyone on the way into work.  
It's a big adjustment not being able to rely on my  
peripheral vision the way I drive. ;)   
[See Wall-to-Wall](#)  
Patrick

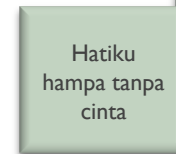
 **Robert Scoble posted a video.**Share +  
  
**Allison Lovejoy plays Chopin**  
by Tom Foremski  
13:48 Uploaded on Sunday  
 Some nice piano. How did I  
find this? Trolling Facebook's  
video feeds.   
[Add a comment](#)

 Corey Hammond is at work...  
 James Avery is attending SW Alumni Drinks in London.  
 Allan J. Cox and Tony Wright joined the group Strumpettes.

# Search System Desiderata

---

- Want a system that:
- Given a query (set of keywords) returns top-k most recent posts containing these keywords
- adheres to the privacy settings of users (can only retrieve friends' posts)
- makes new posts immediately searchable
- answers queries quickly
- does not consume too much space



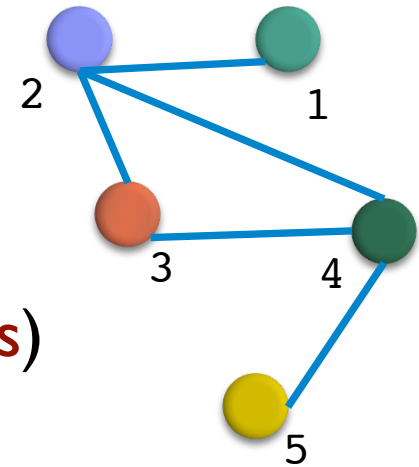
yiippiiii



# Informal Problem Definition

---

- Given a social network
  - Nodes = Users
  - Edges indicate friendship (selflinks omitted)
- Given posts written by users (the **authors**)
- **Answer conjunctive queries**
  - Result of a query
    - Top-k most recent posts that
      1. Contain all keywords of the query
      2. Are authored by friends
  - **Queries with access control**



# Design Space

---

- Two axes of enforcing access control:
  - Index axis:
    - A **group index** contains the posts of a subset of users
    - An **index design** is a set of group indexes
  - Access axis:
    - A **group author list** is a sorted list of pairs  $\langle \text{post-ID}, \text{author-ID} \rangle$  for a subset of users
    - An **access design** is a set of group author lists
  - Intuition: Query processing

# Examples - Index / Access Designs

---

We distinguish designs based on:

|             | Index Design                         | Access Design                      |
|-------------|--------------------------------------|------------------------------------|
| Cardinality | # of indexes                         | # of author lists                  |
| Redundancy  | avg # of indexes a user is member of | avg # of lists a user is member of |

# Examples – Index Designs

## Global Index

► I1:     

No redundancy  
Lowest cardinality

## Friends Indexes:

► I1:  

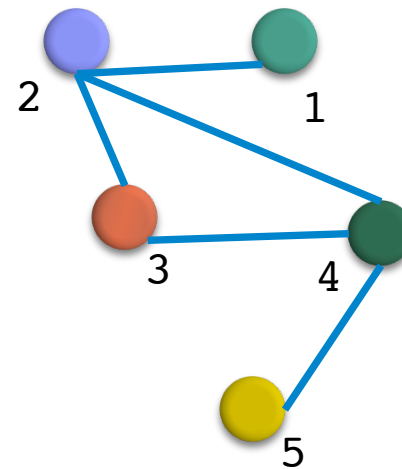
► I2:   

► I3:   

► I4:    


► I5:  

High redundancy  
High cardinality



# Examples – Access Designs

## Global List

► L1 : 

No redundancy  
Lowest cardinality

## Friends List:

► L1 : 

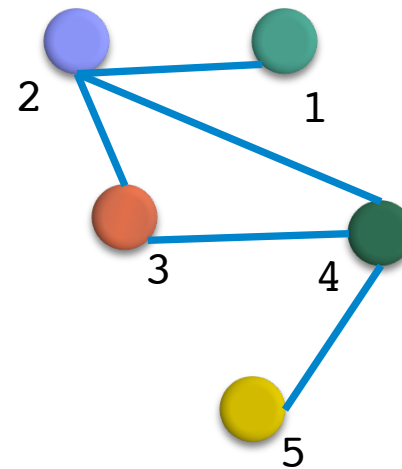
► L2 : 

► L3 : 

► L4 : 

► L5 : 

High redundancy  
High cardinality



# Terminology

---

- **Covers:**
  - A set of *group indexes* **covers** a set of users if each user's posts are contained in at least one *group index*.
    - **Exact covers:** no posts of other users.
  - A set of *group author lists* **covers** a set of users if each user's posts are contained at least one *group author list*.
    - **Exact covers:** no posts of other users.

# Query Processing with Access Control

---

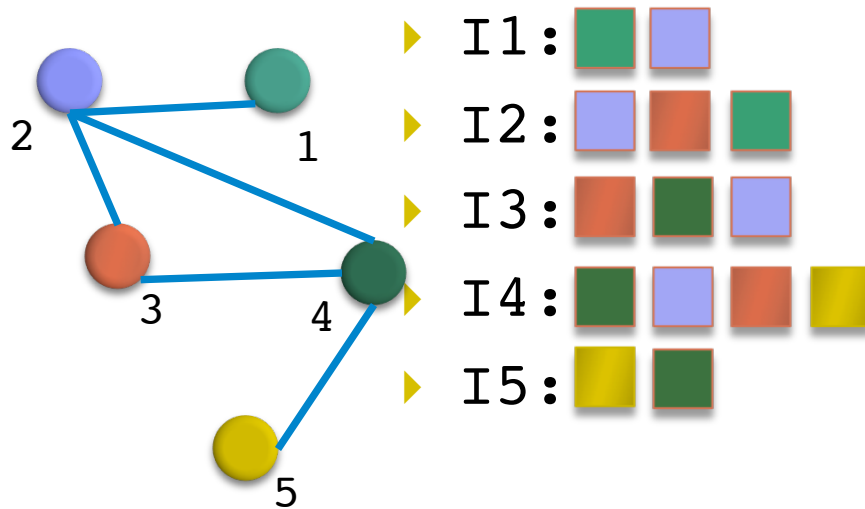
- Given query  $t_1, \dots, t_m$  by user  $u$ 
  1. Select indexes covering  $u$ 's friends.
  2. Select author lists covering  $u$ 's friends.
  3. Within each selected index:
    - a) Intersect posting lists for  $t_1, \dots, t_m$  with the union of the selected author lists.
    - b) For each result check whether  $u$  is friends with author.
  4. Union the results of the indexes

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

- Processing Optimizations:
  - Group indexes are exact cover  $\rightarrow$  no further filtering (skip 2., 3.)
  - Group author lists are exact cover  $\rightarrow$  no friendship check (skip 3.b))

# Examples – Index + Access Designs

## Friends Indexes / No Group Author lists:

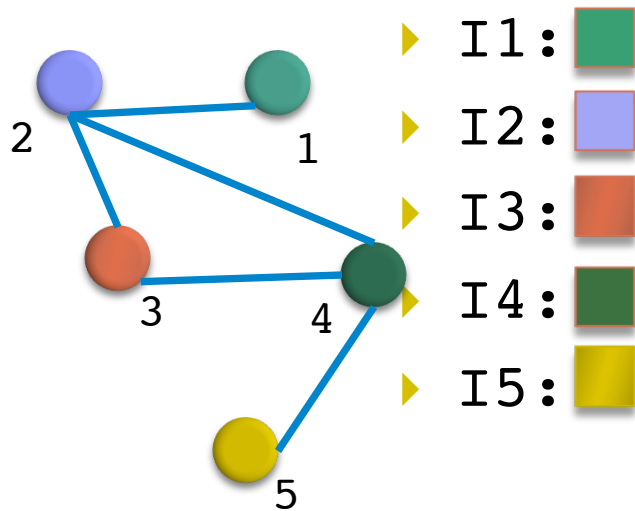


High redundancy  
 High cardinality  
 Optimization: No author lists

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

# Examples - Index + Access Designs

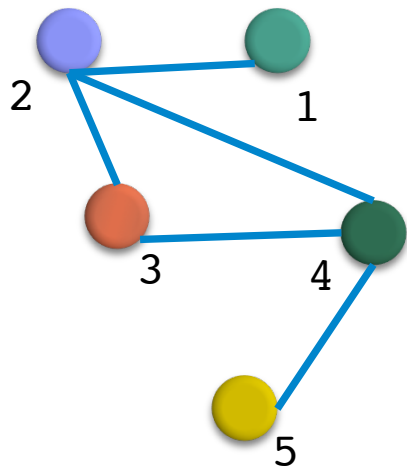
## User Indexes / No Group Author lists:



No redundancy  
 High cardinality  
 Optimization: No friends lookup

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

# Examples – Index + Access Designs




Global index / Global list

**Index:**

► I1: 

No redundancy  
Lowest cardinality

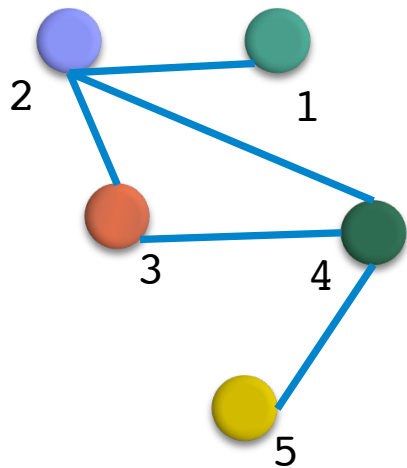
**Author lists:**

► L1: 

No redundancy  
Lowest cardinality

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

# Examples – Index + Access Designs








Global index / User list

**Index:**

▶ I1 :     

No redundancy  
Lowest cardinality

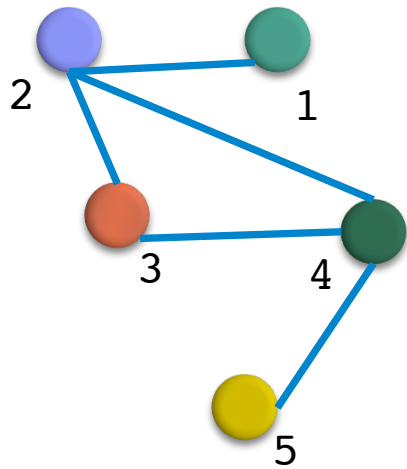
**Author lists:**

▶ L1 :   
 ▶ L2 :   
 ▶ L3 :   
 ▶ L4 :   
 ▶ L5 : 

No redundancy  
High cardinality  
Optimization:  
No friends look-up

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

# Examples – Index + Access Designs



Global index / Friends list

**Index:**

▶ I1:

No redundancy  
Lowest cardinality

**Author lists:**

▶ L1:

▶ L2:

▶ L3:

▶ L4:

▶ L5:

High redundancy  
High cardinality  
Optimization:  
No friends look-up

$$\bigcup_{\text{indexes}} \sigma_{\text{friends}} \left( \left( \bigcap \text{posting lists for } t_1, \dots, t_m \right) \cap \left( \bigcup \text{author lists} \right) \right)$$

# Implementation - Overview

---

- Main memory system in Java
- Updates:
  - Small updatable index to add new posts
  - hierarchy of indexes based on geometric partitioning with compression
- Operators over lists:
  - Operators: Union, Intersection, Filter
  - Methods: `Next()`, `SkipTo(value v)`

# Experiments

---

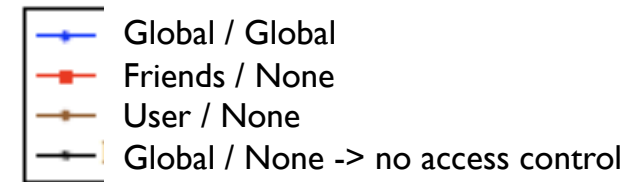
- **Comparison of the performance**
  - Across index designs
  - Across access designs
  - Across different social networks
- **Performance measures:**
  - Time to answer query
  - Time to add post
  - Space consumption

# Experiments

---

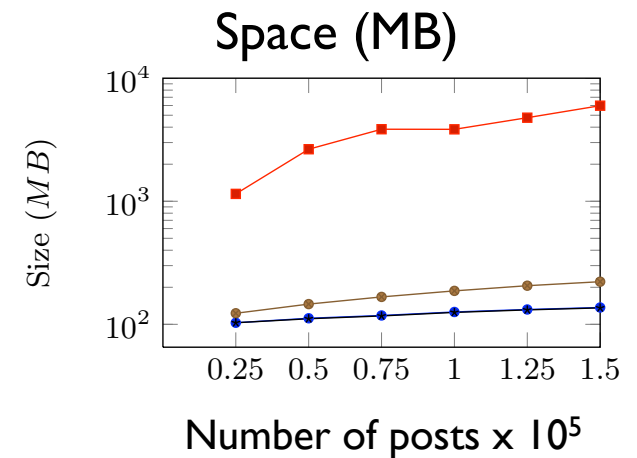
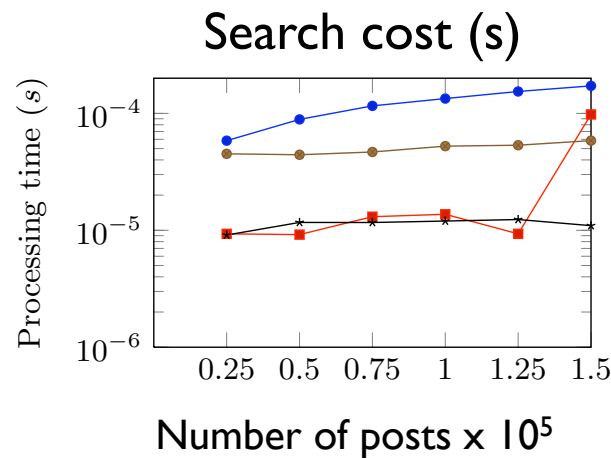
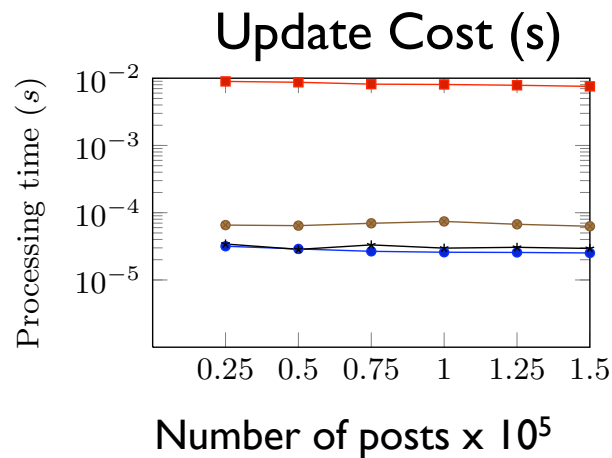
- Data:
  - Network:
    - real twitter network: 417,000 users
    - synthetic networks (Barabasi's attachment model) varying size and degree
  - Posts obtained from twitter
  - Queries
    - generated through a random process
    - Run 100,000 queries returning top-100 posts
- Environment: 3.2GHz, 16GB RAM, Red Hat Enterprise 5.3

# Scalability

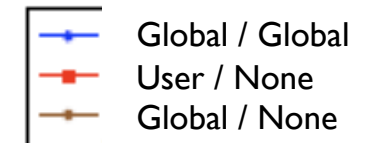


## Varying the number of documents

fix 1,000 users, 20 friends per user



# Index Design Performance under Varying Networks



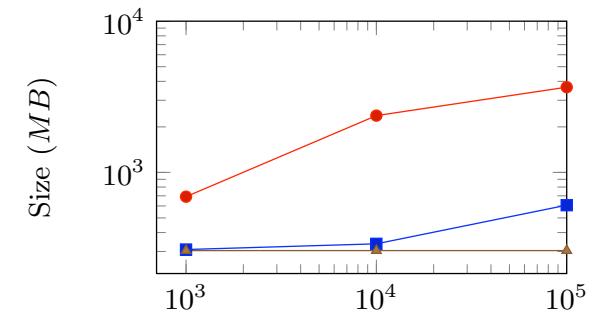
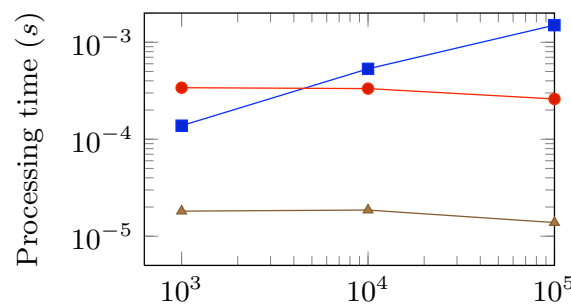
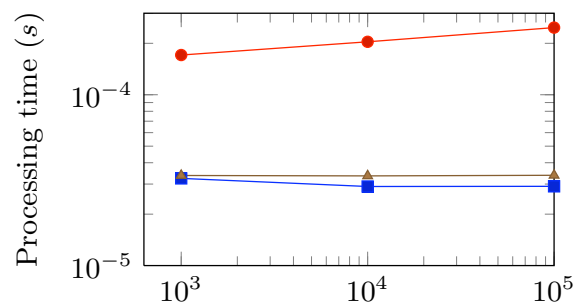
Update Cost (s)

Search cost (s)

Space (MB)

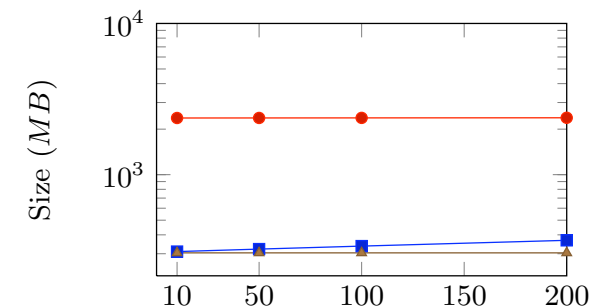
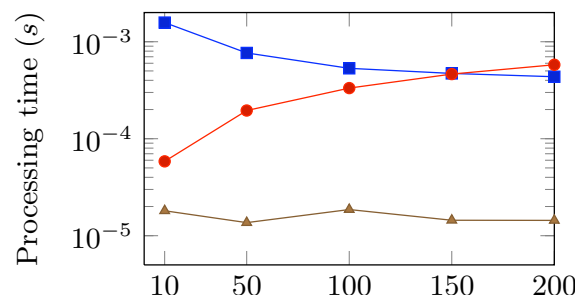
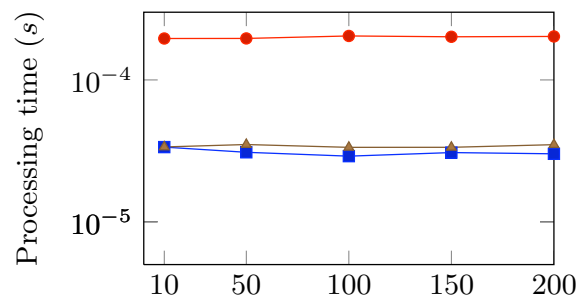
Varying the number of users

fix 100 friends per user, 1mm posts



Varying the number of friends per user

fix 10,00 users, 1mm posts



# Access Design Performance under Varying Networks



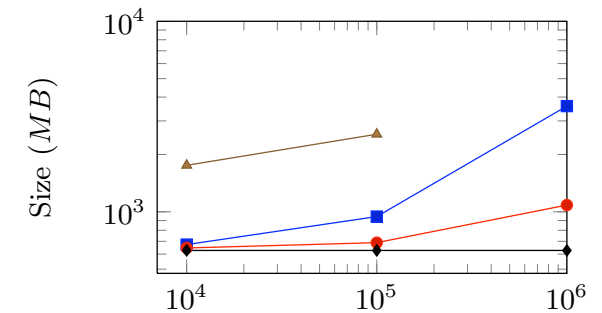
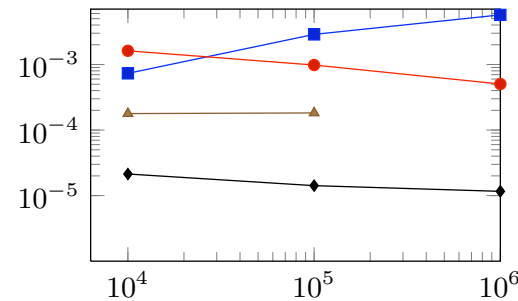
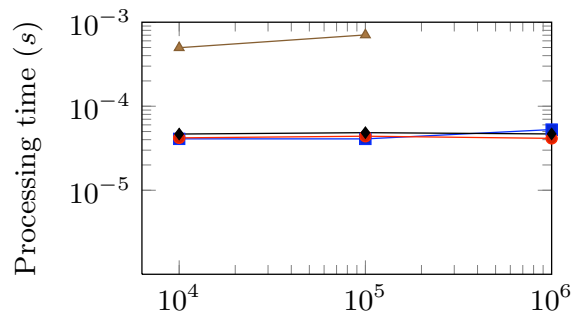
Update Cost (s)

Search cost (s)

Space (MB)

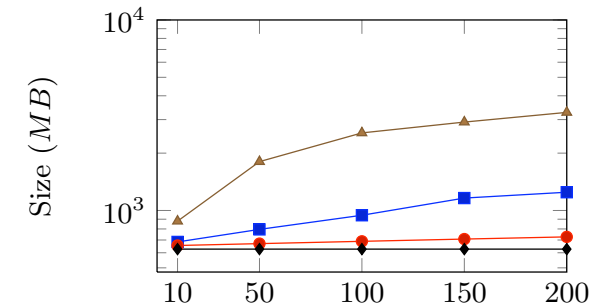
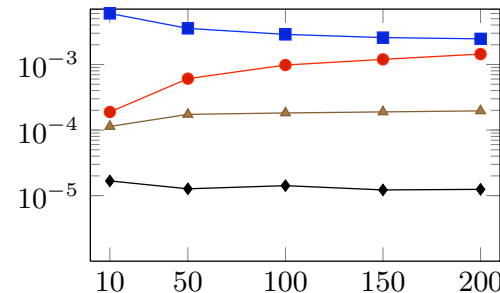
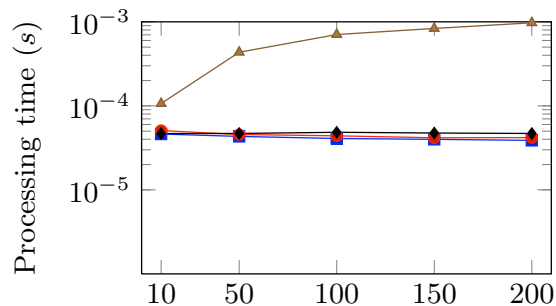
**Varying the number of users**

fix 100 friends per user, 2.5mm posts



**Varying the number of friends per user**

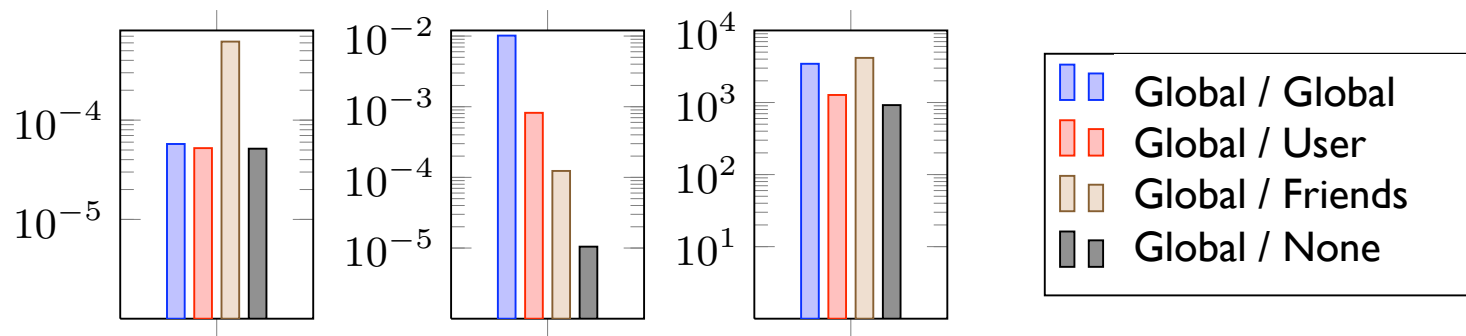
fix 100,00 users, 2.5mm posts



# Experiment on Real Twitter Network

## Access designs with global index on real network

Update Cost (s)   Search cost (s)   Space (MB)



# Conclusions

---

- Two axis design space for access control in search:
  - Index Axis
  - Access Axis
- **Experiments** with five designs:
  - Access designs reveal tradeoffs between index size, update and search performance
    - Global Index / Friends lists
      - fast searches (independent on network)
      - slow updates (dependent on network)
    - Global Index / User lists or Global Index / Global list
      - slow searches (dependent on network)
      - fast updates (independent on network)
  - Similar tradeoffs for index designs
- **Recommendation:** Choose between user indexes and the global index with user or friends lists based on workload and network

## Future Work

---

- Explore design space
  - Identify **best design** for a particular workload and network
- Dynamic design
  - Adapt to **changes in the workload**
  - Adapt to **changes in network**
- Distribute system
- Extend to more advanced ranking functions
  - Include network structure and interactions as features
  - Do not leak private information through ranking

---

Thank You!  
Questions?  
[goetz@cs.cornell.edu](mailto:goetz@cs.cornell.edu)