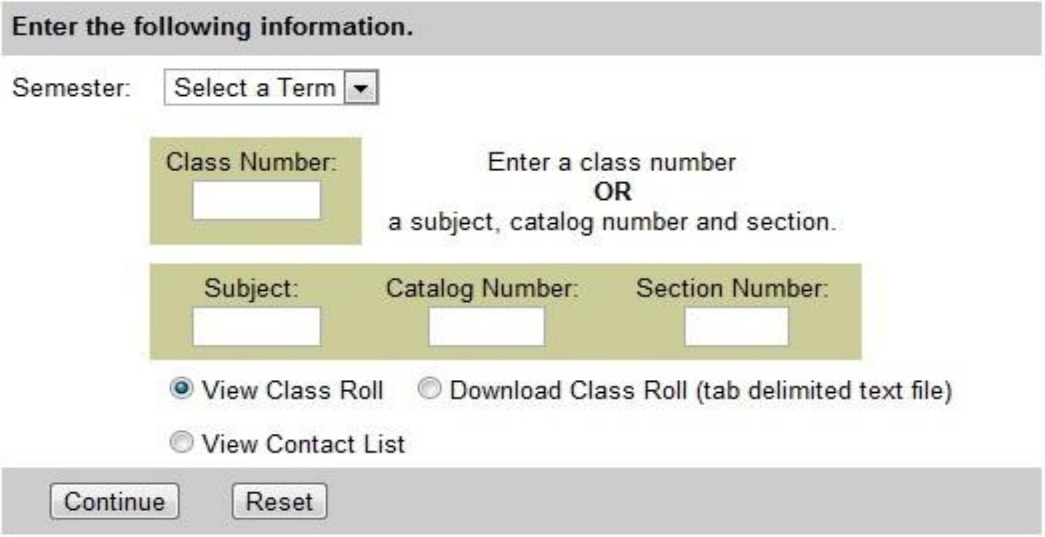
Building Local Search Engines for Big Heterogeneous Data

Cody Hansen, Feifei Li

The Motivation



Enter the following information.

Semester: Select a Term

Class Number:

Enter a class number
OR
a subject, catalog number and section.

Subject: Catalog Number: Section Number:

● View Class Roll ○ Download Class Roll (tab delimited text file)

○ View Contact List

Continue Reset

- Typical search interface:
  - Schema-specific query forms
  - Rigid schema and formats required for the underlying data
  - Each form requires a corresponding program
  - Not very user friendly
    - Many inputs?
    - Domain values?

The Objective

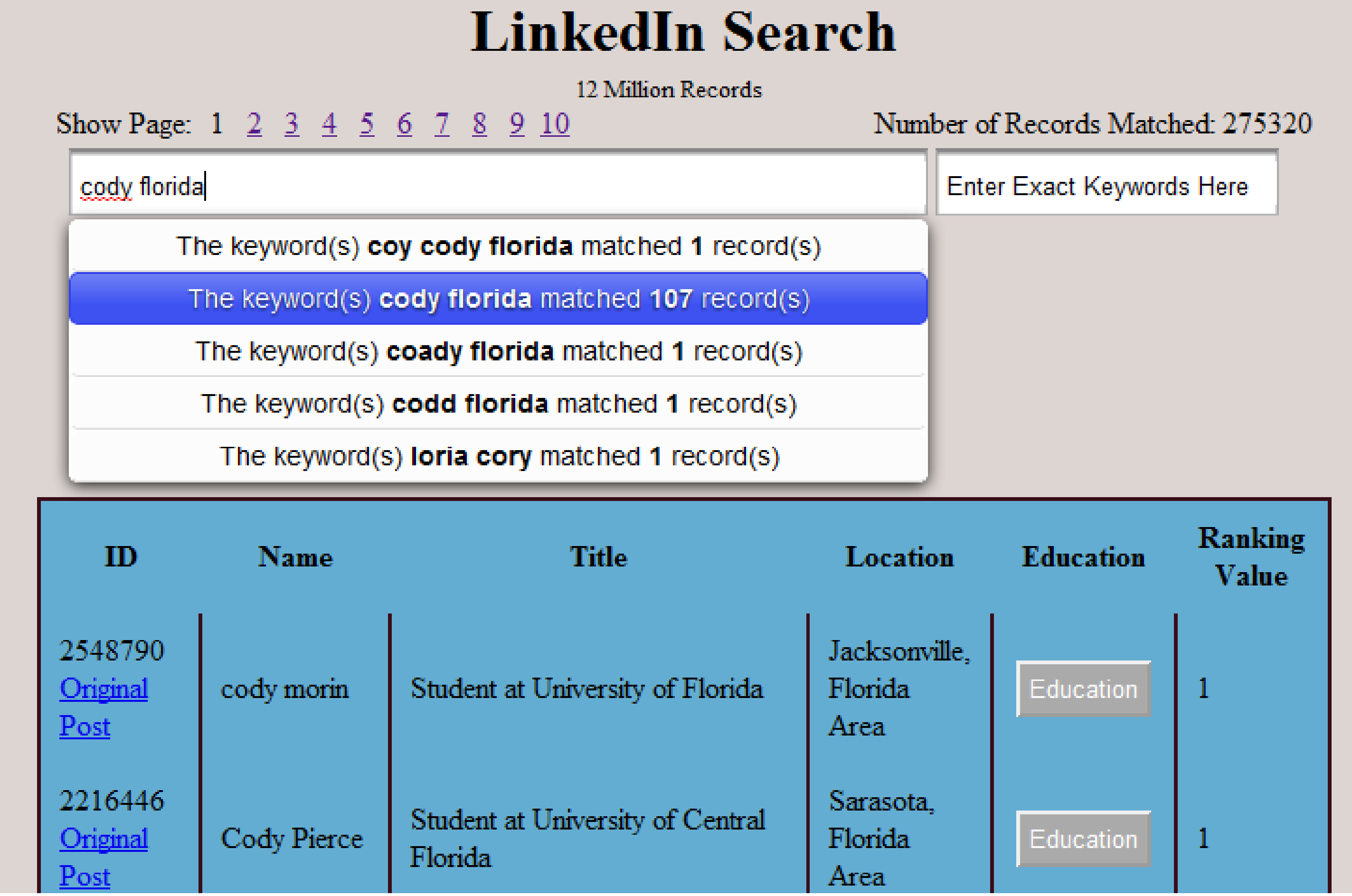
- The objective: a search-engine-style integration, search, ranking, and recommendation system:
  - must handle heterogeneous data sources
  - it is desired to be schemaless and formatless
  - easy to use and flexible search, ranking, and recommendation interface

The Challenges

- How to achieve both efficiency and effectiveness in scale?
  - the big data challenge
  - return useful and meaningful results, as well as effective rankings and recommendations
- Must handle millions of records, or even billions of them, in hundreds of gigabytes or even terabytes

The Search Module

- A search-engine-style approach:



LinkedIn Search

12 Million Records

Show Page: 1 2 3 4 5 6 7 8 9 10

Number of Records Matched: 275320

cody florida

Enter Exact Keywords Here

The keyword(s) **coy cody florida** matched **1** record(s)

The keyword(s) **cody florida** matched **107** record(s)

The keyword(s) **coady florida** matched **1** record(s)

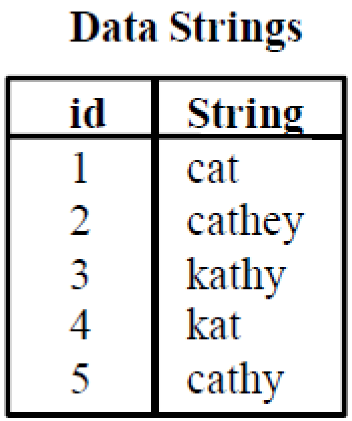
The keyword(s) **codd florida** matched **1** record(s)

The keyword(s) **loria cory** matched **1** record(s)

ID	Name	Title	Location	Education	Ranking Value
2548790 Original Post	cody morin	Student at University of Florida	Jacksonville, Florida Area	Education	1
2216446 Original Post	Cody Pierce	Student at University of Central Florida	Sarasota, Florida Area	Education	1

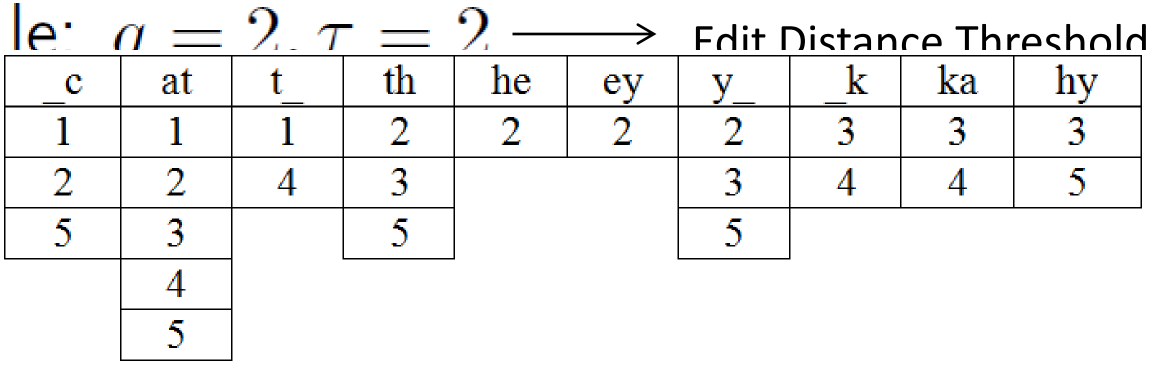
Basic Idea

- A keyword-centric approach
  - Regardless of data types, each attribute is parsed into a set of keywords
  - Inverted lists to index these keywords (keyword to record ids), with our own storage engine
  - Another set of inverted lists to index q-grams to keywords (for approximate keyword matching)



Data Strings

id	String
1	cat
2	cathey
3	kathy
4	kat
5	cathy

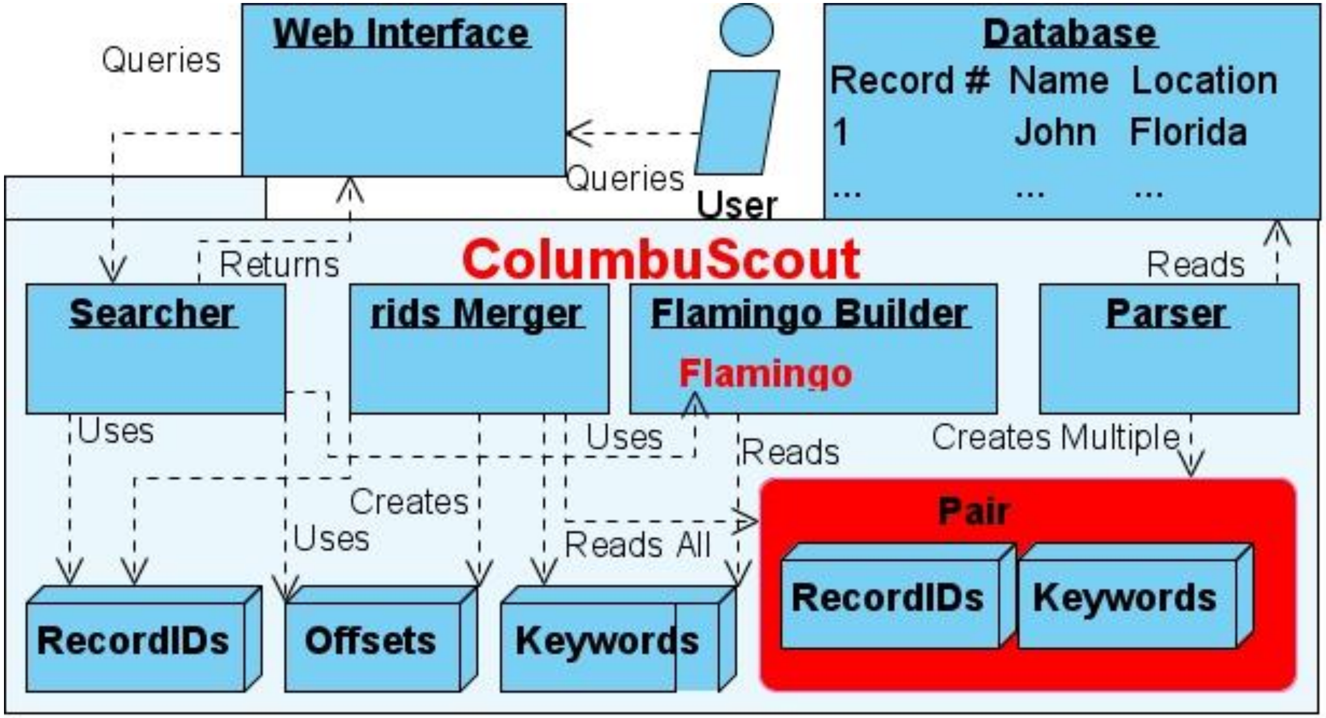


le: $q = 2, \tau = 2$ ⟶ Edit Distance Threshold

_c	at	t_	th	he	ey	y_	_k	ka	hy
1	1	1	2	2	2	2	3	3	3
2	2	4	3			3	4	4	5
5	3		5			5			
	4								
	5								

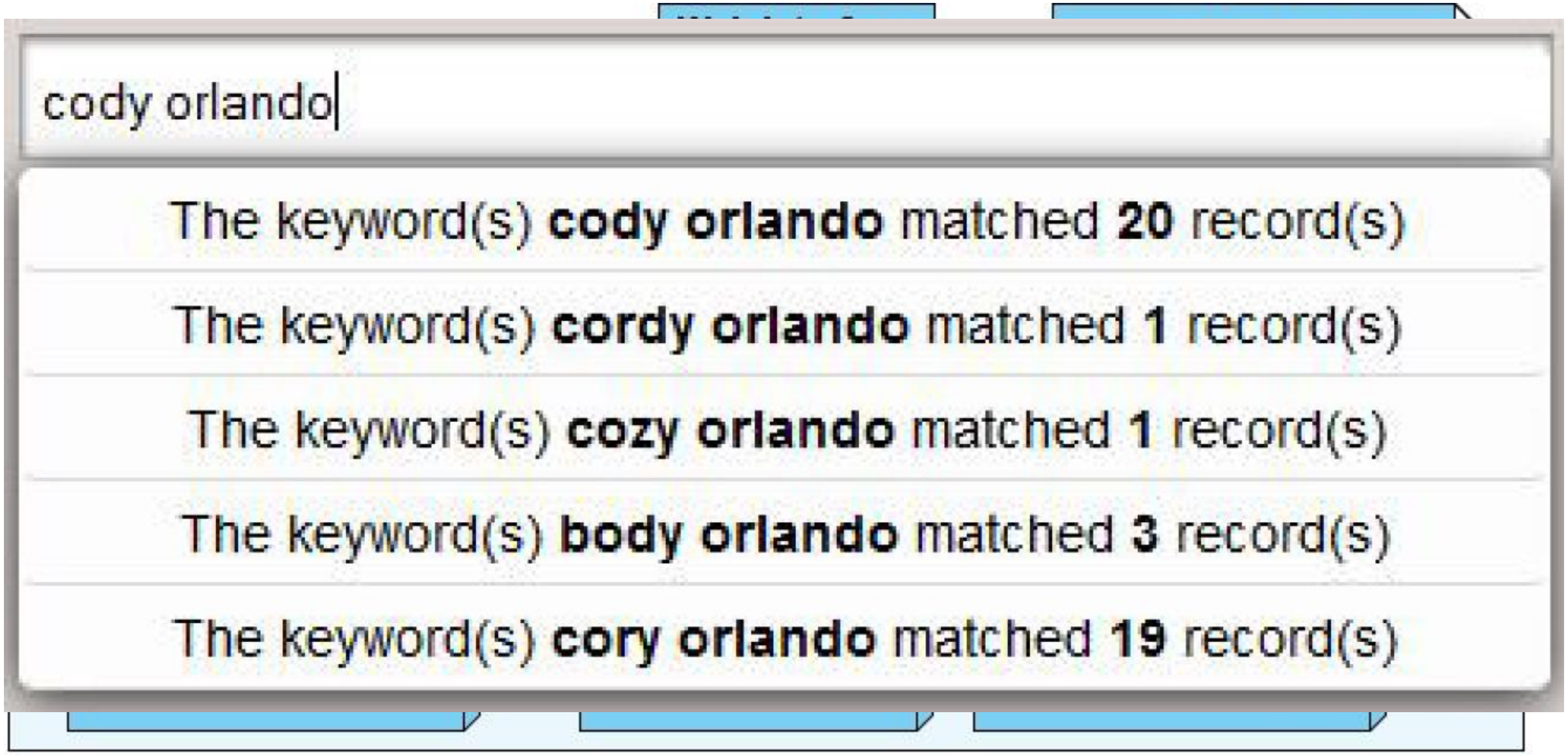
System Architecture

- Main modules: parser, merger (to handle big data), flamingo builder, searcher



Searcher

- The searcher has the following main steps:
  - Find approximate keywords
  - Find RIDs
  - Merge them
  - Make Recommendations and Rankings



cody orlando

The keyword(s) **cody orlando** matched **20** record(s)

The keyword(s) **cordy orlando** matched **1** record(s)

The keyword(s) **cozy orlando** matched **1** record(s)

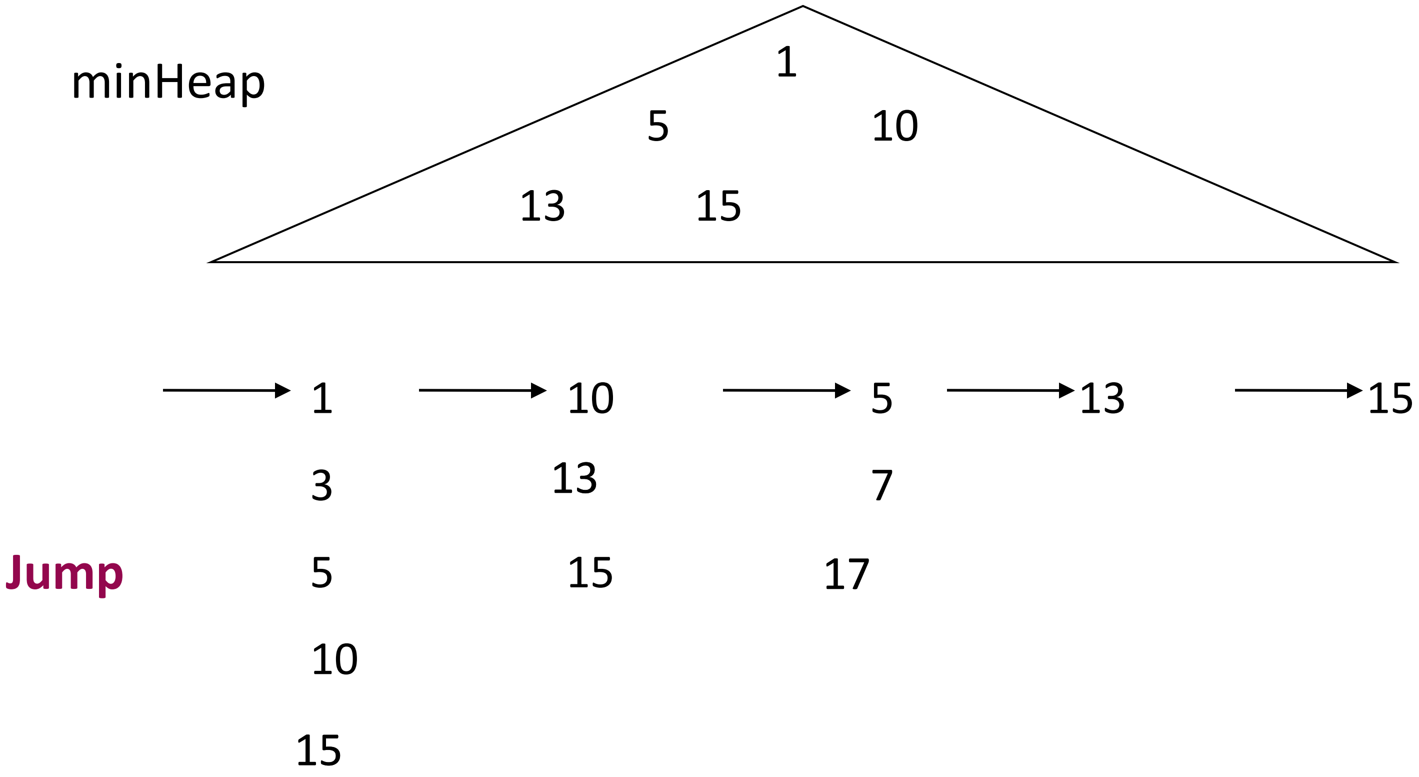
The keyword(s) **body orlando** matched **3** record(s)

The keyword(s) **cory orlando** matched **19** record(s)

Merger

- MergeSkip algorithm designed for q-gram merging.
- Basic idea is keep a pointer in each list.
- When you fail an ID, do a binary search for the next number in each of the lists

Example of MergeSkip



minHeap

1

5 10

13 15

	1	10	5	13	15
	3	13	7		
Jump	5	15	17		
	10				
	15				

Count threshold T≥ 4

Other Features

- Also support
  - Column specific search: column = keyword, or column = “keyword1 keyword2 …”
  - Exact search: exact = keyword (search anywhere), or column == keyword (search on that column)
  - Can combine them in anyway, e.g.,

cody title = “stdent florida” tallahssee education == state exact = hansen

cody, tallahssee: approximate search anywhere

stdent florida: approx search on title

state: exact search on education

hansen: exact search anywhere

Other Issues

- How to achieve effective ranking and recommendation?
  - TF-IDF style approach
  - Associations
  - Ontology
- How to build the indices and storage engine extremely fast and scalable?
  - Use MapReduce to do this in parallel
- Use a cluster of commodity machines for search as well?
- How to handle streaming updates efficiently?

Associations

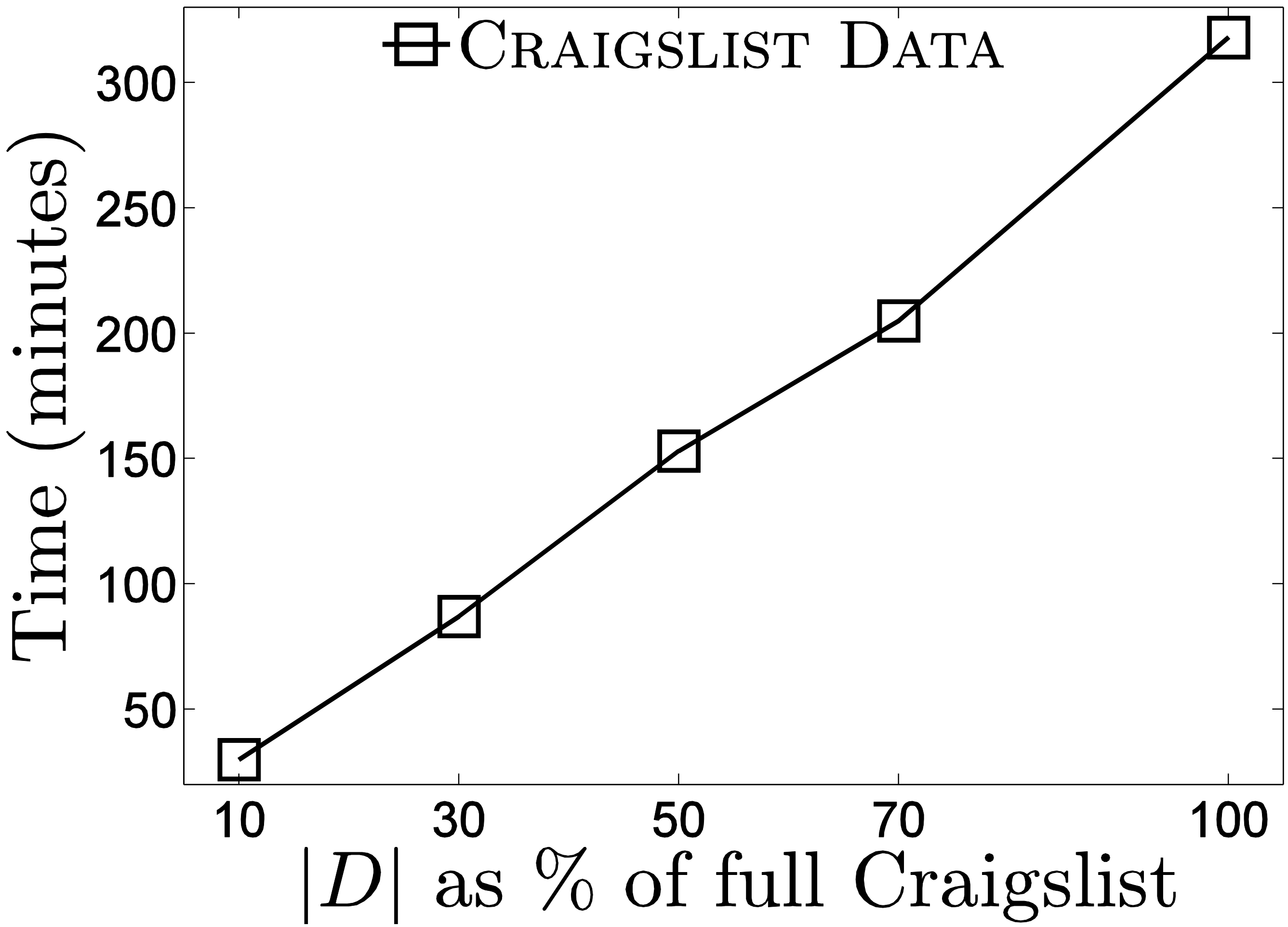
- Goal: Find the words that appear together at least T times.

TID	Keywords
1	1 3 4
2	2 3 5
3	1 2 3 5
4	2 5

Results

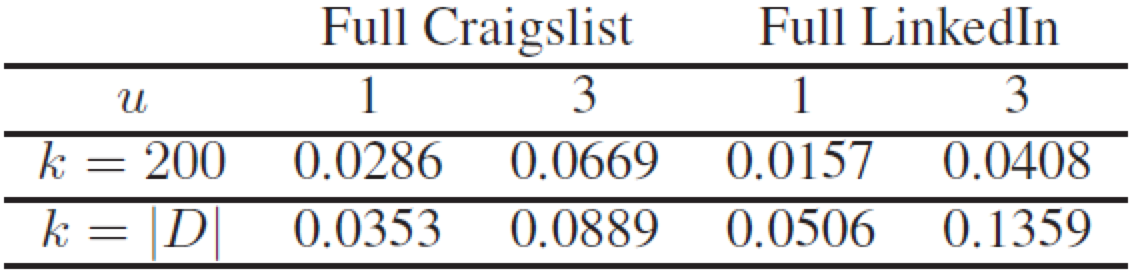
- Craiglist data: 1.7 billion records, 300GB.
- LinkedIn data: 12 million records, 10GB.
- A few Million unique keywords
- A single linux machine running ubuntu 12.9 and mysql server 5.1, with 12GB ram, 2TB disk, and a single Intel ®CPU X3470@2.93GHz

Results (continued)



Results (continued)

- u: number of keywords searched
- k: number of recommendations made
- Query efficiency in second:



	Full Craigslist		Full LinkedIn	
u	1	3	1	3
$k = 200$	0.0286	0.0669	0.0157	0.0408
$k = \|D\|$	0.0353	0.0889	0.0506	0.1359

A live demo

http://datagroup.cs.utah.edu/columbuscout.php