Large-scale social media analysis with Hadoop

Jake Hofman
Yahoo! Research
May 23, 2010
• Few direct observations; highly detailed info on nodes and edges
• E.g. karate club (Zachary, 1977)

The karate club was observed for a period of three years, from 1970 to 1972. In addition to direct observation, the history of the club prior to the period of the study was reconstructed through informants and club records in the university archives. During the period of observation, the...
1990s $\sim 10^4$ nodes

- Larger, indirect samples; relatively few details on nodes and edges
- E.g. APS co-authorship network (http://bit.ly/aps08jmh)
Very large, dynamic samples; many details in node and edge metadata

E.g. Mail, Messenger, Facebook, Twitter, etc.
If you had all of twitter's data, what question would you ask of it?

djw2451
If you had all of twitter's data, what question would you ask of it?

What *could* you ask of it?
Look familiar?

```
# ls -talh neat_dataset.tar.gz
-rw-r--r-- 100T May 23 13:00 neat_dataset.tar.gz
```
Look familiar?¹

```
# ls -talh twitter_rv.tar
-rw-r--r--  24G May 23 13:00 twitter_rv.tar
```

¹http://an.kaist.ac.kr/traces/WWW2010.html
Large-scale social media analysis with Hadoop
Agenda

Large-scale social media analysis with Hadoop

GB/TB/PB-scale, 10,000+ nodes
Large-scale social media analysis with Hadoop

network & text data
Large-scale social media analysis with Hadoop

network analysis & machine learning
Large-scale social media analysis with Hadoop

open source Apache project for distributed storage/computation
Warning

You may be bored if you already know how to ...
Warning

You may be bored if you already know how to ...

- Install and use Hadoop (on a single machine and EC2)
Warning

You may be bored if you already know how to ...

- Install and use Hadoop (on a single machine and EC2)
- Run jobs in local and distributed modes
Warning

You may be bored if you already know how to ... 

- Install and use Hadoop (on a single machine and EC2)
- Run jobs in local and distributed modes
- Implement distributed solutions for:
  - Parsing and manipulating large text collections
  - Clustering coefficient, BFS, etc., for networks with billions of edges
  - Classification, clustering
Warning

You may be bored if you already know how to ...

- Install and use Hadoop (on a single machine and EC2)
- Run jobs in local and distributed modes
- Implement distributed solutions for:
  - Parsing and manipulating large text collections

@jakehofman (Yahoo! Research)
Warning

You may be bored if you already know how to ...

- Install and use Hadoop (on a single machine and EC2)
- Run jobs in local and distributed modes
- Implement distributed solutions for:
  - Parsing and manipulating large text collections
  - Clustering coefficient, BFS, etc., for networks w/ billions of edges
Warning

You may be bored if you already know how to ...

- Install and use Hadoop (on a single machine and EC2)
- Run jobs in local and distributed modes
- Implement distributed solutions for:
  - Parsing and manipulating large text collections
  - Clustering coefficient, BFS, etc., for networks w/ billions of edges
  - Classification, clustering
Selected resources

http://www.hadoopbook.com/
Selected resources

http://www.cloudera.com/
Selected resources

Data-Intensive Text Processing with MapReduce

Jimmy Lin and Chris Dyer
University of Maryland, College Park

Selected resources

... and many more at

http://delicious.com/jhofman/hadoop
Selected resources

... and *many* more at

http://delicious.com/pskomoroch/hadoop
Outline

1. Background (5 Ws)

2. Introduction to MapReduce (How, Part I)

3. Applications (How, Part II)
What?
“... to create building blocks for programmers who just happen to have lots of data to store, lots of data to analyze, or lots of machines to coordinate, and who don’t have the time, the skill, or the inclination to become distributed systems experts to build the infrastructure to handle it.”

-Tom White

_Hadoop: The Definitive Guide_
What?

Hadoop contains many subprojects:

- **Hadoop Common**: The common utilities that support the other Hadoop subprojects.
- **Chukwa**: A data collection system for managing large distributed systems.
- **HBase**: A scalable, distributed database that supports structured data storage for large tables.
- **HDFS**: A distributed file system that provides high throughput access to application data.
- **Hive**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **MapReduce**: A software framework for distributed processing of large data sets on compute clusters.
- **Pig**: A high-level data-flow language and execution framework for parallel computation.
- **ZooKeeper**: A high-performance coordination service for distributed applications.

We’ll focus on distributed computation with **MapReduce**.
Who/when?

An overly brief history
Who/when?

**pre-2004**

Cutting and Cafarella develop open source projects for web-scale indexing, crawling, and search

[Lucene](https://lucene.apache.org/)  
[##nutch](https://nutch.apache.org/)
Who/when?

2004

Dean and Ghemawat publish MapReduce programming model, used internally at Google

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.
2006
Hadoop becomes official Apache project, Cutting joins Yahoo!, Yahoo adopts Hadoop
Who/when?

hadoop

Search Volume index
6.00
4.00
2.00
0

2004  2005  2006  2007  2008  2009  2010

News reference volume
0

Google Trends
A
B
C
E

Large-scale social media analysis w/ Hadoop
Where?

http://wiki.apache.org/hadoop/PoweredBy
Why yet another solution?

(I already use too many languages/environments)
Why?

Why a *distributed* solution?

(My desktop has TBs of storage and GBs of memory)
Roughly how long to read 1TB from a commodity hard disk?
Roughly how long to read 1TB from a commodity hard disk?

\[
\frac{1 \text{ Gb}}{2 \text{ sec}} \times \frac{1 \text{ B}}{8 \text{ b}} \times \frac{3600 \text{ sec}}{\text{hr}} \approx 225 \frac{\text{GB}}{\text{hr}}
\]
Why?

Roughly how long to read 1TB from a commodity hard disk?

\[ \approx 4\text{hrs} \]
Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds

We used Apache Hadoop to compete in Jim Gray's Sort benchmark. Jim's Gray's sort benchmark consists of a set of many related benchmarks, each with their own rules. All of the sort benchmarks measure the time to sort different numbers of 100 byte records. The first 10 bytes of each record is the key and the rest is the value. The minute sort must finish end to end in less than a minute. The Gray sort must sort more than 100 terabytes and must run for at least an hour. The best times we observed were:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Nodes</th>
<th>Maps</th>
<th>Reduces</th>
<th>Replication</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000,000,000,000</td>
<td>1406</td>
<td>8000</td>
<td>2600</td>
<td>1</td>
<td>59 seconds</td>
</tr>
<tr>
<td>1,000,000,000,000,000</td>
<td>1460</td>
<td>8000</td>
<td>2700</td>
<td>1</td>
<td>62 seconds</td>
</tr>
<tr>
<td>100,000,000,000,000,000</td>
<td>3452</td>
<td>190,000</td>
<td>10,000</td>
<td>2</td>
<td>173 minutes</td>
</tr>
<tr>
<td>1,000,000,000,000,000,000</td>
<td>3658</td>
<td>80,000</td>
<td>20,000</td>
<td>2</td>
<td>975 minutes</td>
</tr>
</tbody>
</table>

Outline

1 Background (5 Ws)

2 Introduction to MapReduce (How, Part I)

3 Applications (How, Part II)
Store, parse, and analyze high-volume server logs,

e.g. how many search queries match “icwsm”?
Break large problem into smaller parts, solve in parallel, combine results
Typical scenario

“Embarassingly parallel”
(or nearly so)
How many search queries match “icwsm”, grouped by month?
MapReduce: example

Map
matching records to
(YYYYMM, count=1)

Shuffle
to collect all records
w/ same key (month)

Reduce
results by adding
count values for each key

20091201, 4.2.2.1, "icwsm 2010"
20100523, 2.4.1.2, "hadoop"
20100101, 9.7.6.5, "tutorial"
20091125, 2.4.6.1, "data"
20090708, 4.2.2.1, "open source"
20100124, 1.2.2.4, "washington dc"

20100522, 2.4.1.2, "conference"
20091008, 4.2.2.1, "2009 icwsm"
20090807, 4.2.2.1, "apache.org"
20100101, 9.7.6.5, "mapreduce"
20100123, 1.2.2.4, "washington dc"
20091121, 2.4.6.1, "icwsm dates"

20090807, 4.2.2.1, "distributed"
20091225, 4.2.2.1, "icwsm"
20100522, 2.4.1.2, "media"
20100123, 1.2.2.4, "social"
20091114, 2.4.6.1, "d.c."
20100101, 9.7.6.5, "new year's"

200912, 1
200912, 1

200910, 1
200911, 1

200912, 1
200912, 1

200910, 1
200910, 1

200912, 2
...

200911, 1
...

jakehofman (Yahoo! Research)

Large-scale social media analysis w/ Hadoop

May 23, 2010 28 / 71
MapReduce: paradigm

Programmer specifies map and reduce functions
MapReduce: paradigm

**Map**: transforms input record to intermediate (key, value) pair

```python
def mapper(record):
    # input: a single record

    # parse / transform / filter record
    ...

    # output: intermediate key(s) and value(s)
    output((key, value))
```
MapReduce: paradigm

Shuffle: collects all intermediate records by key
Reduce: transforms all records for given key to final output

```python
def reducer(key, records):
    # input: intermediate key and all values
    # initialize variables, e.g. counters
    for record in records:
        # parse record
        ...  
        # update variables, e.g. count
    # output: final key(s) and value(s)
    output((key, value))
```
MapReduce: paradigm

Distributed read, shuffle, and write are transparent to programmer
MapReduce: principles

- Move code to data (local computation)
- Allow programs to scale transparently w.r.t size of input
- Abstract away fault tolerance, synchronization, etc.
MapReduce: strengths

- Batch, offline jobs
- Write-once, read-many across full data set
- Usually, though not always, simple computations
- I/O bound by disk/network bandwidth
What it’s not:

- High-performance parallel computing, e.g. MPI
- Low-latency random access relational database
- Always the right solution
the quick brown fox
jumps over the lazy dog
who jumped over that
lazy dog -- the fox?
Map: for each line, output each word and count (of 1)

the quick brown fox
jumps over the lazy dog
who jumped over that
lazy dog -- the fox?
Shuffle: collect all records for each word

the quick brown fox
jumps over the lazy dog
who jumped over that
lazy dog -- the fox?
Reduce: add counts for each word

```
-- 1
---------
? 1
---------
brown 1
dog 1
dog 1
---------
fox 1
fox 1
---------
jumped 1
---------
jumps 1
---------
lazy 1
lazy 1
---------
over 1
over 1
---------
quick 1
---------
that 1
---------
the 1
the 1
the 1
---------
who 1
```
the quick brown fox

jumps over the lazy dog

who jumped over that

lazy dog -- the fox?

@jakehofman (Yahoo! Research) Large-scale social media analysis w/ Hadoop May 23, 2010 33 / 71
package org.apache;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.mapreduce.lib.partition.*;
import org.apache.hadoop.mapreduce.lib.input.CombineFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.CombineFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.CombineInputFormat;
import org.apache.hadoop.mapreduce.lib.output.CombineOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.CombineInputFormat;
Hadoop streaming

Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run map/reduce jobs with any executable or script as the mapper and/or the reducer. For example:

```bash
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /bin/wc
```
Hadoop streaming

MapReduce for *nix geeks\(^2\):

\[\text{# cat data } | \text{ map } | \text{ sort } | \text{ reduce}\]

- Mapper reads input data from stdin
- Mapper writes output to stdout
- Reducer receives input, sorted by key, on stdin
- Reducer writes output to stdout

\(^2\text{http://bit.ly/michaelnoll}\)
Locally:

```
# cat data | tr " " "\n" | sort | uniq -c
```

wordcount.sh

Locally:

```
# cat data | tr " " "\n" | sort | uniq -c
```

Distributed:

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
  -input README.txt \
  -output wordcount \
  -mapper 'tr " " "\n" ' \
  -reducer 'uniq -c'
```
Use the same code on MBs locally or TBs across thousands of machines.
from hstream import HStream
import sys
import re
from collections import defaultdict

class WordCount(HStream):
    
def mapper(self, record):
        for word in " ".join(record).split():
            self.write_output((word, 1))

    def reducer(self, key, records):
        total = 0
        for record in records:
            word, count = record
            total += int(count)
        self.write_output((word, total))

    if __name__ == '__main__':
        WordCount()
Outline

1. Background (5 Ws)

2. Introduction to MapReduce (How, Part I)

3. Applications (How, Part II)
Network data
Scale

- Example numbers:
  - $\sim 10^7$ nodes
  - $\sim 10^2$ edges/node
  - no node/edge data
  - static
  - $\sim 8$GB

![Diagram of user network](image)
Example numbers:
- $\sim 10^7$ nodes
- $\sim 10^2$ edges/node
- no node/edge data
- static
- $\sim 8$GB

Simple, static networks push memory limit for commodity machines
Large-scale social media analysis w/ Hadoop
Example numbers:
- \( \sim 10^7 \) nodes
- \( \sim 10^2 \) edges/node
- node/edge metadata
- dynamic
- \( \sim 100 \text{GB}/\text{day} \)
Scale

- Example numbers:
  - $\sim 10^7$ nodes
  - $\sim 10^2$ edges/node
  - node/edge metadata
  - dynamic
  - $\sim 100$GB/day

Dynamic, data-rich social networks exceed memory limits; require considerable storage
Assumptions

Look only at topology, ignoring node and edge metadata
**Assumptions**

Full network exceeds memory of single machine

\[ A = \]

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[ \cdots \]
Assumptions

Full network exceeds memory of single machine

$A =$

$\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}$
Assumptions

First-hop neighborhood of any individual node fits in memory
Distributed network analysis

MapReduce convenient for parallelizing individual node/edge-level calculations
Distributed network analysis

Higher-order calculations more difficult, but can be adapted to MapReduce framework
Distributed network analysis

- Network creation/manipulation
  - Logs $\rightarrow$ edges
  - Edge list $\leftrightarrow$ adjacency list
  - Directed $\leftrightarrow$ undirected
  - Edge thresholds

- First-order descriptive statistics
  - Number of nodes
  - Number of edges
  - Node degrees

- Higher-order node-level descriptive statistics
  - Clustering coefficient
  - Implicit degree
  - ...

- Global calculations
  - Pairwise connectivity
  - Connected components
  - Minimum spanning tree
  - Breadth-first search
  - Pagerank
  - Community detection
Edge list → adjacency list
Edge list → adjacency list

source  target
1  4
1  5
1  6
7  6
7  1
3  1
1  3
4  2
3  2
2  8
10  2
2  9
3  4
4  3

node  in/out neighbors
1  3 7 3 5 4 6
10  2
2  10 3 4 9 8
3  1 4 1 2 4
4  1 3 3 2
5  1
6  1 7
7  1 6
8  2
9  2
Edge list → adjacency list

Map: for each (source, target), output (source, →, target) & (target, ←, source)

```
node  direction  neighbor
1    >        4
4    <        1
----------
1    >        5
5    <        1
----------
1    >        6
6    <        1
----------
7    >        6
6    <        7
----------
7    >        1
1    <        7
----------
3    >        1
1    <        3
----------
1    >        3
3    <        1
----------
```

source  target
1        4
--------
1        5
--------
1        6
--------
7        6
--------
7        1
--------
3        1
--------
1        3
--------
Edge list → adjacency list

Shuffle: collect each node’s records

<table>
<thead>
<tr>
<th>node</th>
<th>direction</th>
<th>neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>&lt;</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>&gt;</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>&gt;</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>&gt;</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>&gt;</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>&gt;</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>&lt;</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>&lt;</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>&lt;</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>&gt;</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>&gt;</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&lt;</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>&gt;</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>&gt;</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>&gt;</td>
<td>4</td>
</tr>
</tbody>
</table>

source target

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Edge list → adjacency list

Reduce: for each node, concatenate all in- and out-neighbors

```
node  direction  neighbor
1    <          3
1    <          7
1    >          3
1    >          4
1    >          5
1    >          6
-----------------
10    >          2
-----------------
2    <          10
2    <          3
2    <          4
2    >          8
2    >          9
-----------------
3    <          1
3    <          4
3    >          1
3    >          2
3    >          4
```

```
node  in/out neighbors
1    3 7 3 5 4 6
10    2
10    3 4 9 8
3    1 4 1 2 4
4    1 3 3 2
5    1
6    1 7
7    1 6
8    2
9    2
...```

Reduction process:
- Collect all in- and out-neighbors for each node.
- Concatenate these lists to form the adjacency list.
Edge list → adjacency list

node  direction neighbor
1  <  3
1  <  7
1  >  3
1  >  4
1  >  5
1  >  6

source  target

1  4
1  5
1  6
7  6
7  1
3  1
1  3
...

node  in/out neighbors
1  3 7 3 5 4 6
10  2
2  10 3 4 9 8
3  1 4 1 2 4
4  1 3 3 2
5  1
6  1 7
7  1 6
8  2
9  2

@jakehofman  (Yahoo! Research)
Large-scale social media analysis w/ Hadoop

May 23, 2010  52 / 71
Adjacency lists provide access to a node’s *local structure* — e.g. we can *pass messages* from a node to its neighbors.
Degree distribution

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 4</td>
</tr>
<tr>
<td>10</td>
<td>0 1</td>
</tr>
<tr>
<td>2</td>
<td>3 2</td>
</tr>
<tr>
<td>3</td>
<td>2 3</td>
</tr>
<tr>
<td>4</td>
<td>2 2</td>
</tr>
<tr>
<td>5</td>
<td>1 0</td>
</tr>
<tr>
<td>6</td>
<td>2 0</td>
</tr>
<tr>
<td>7</td>
<td>0 2</td>
</tr>
<tr>
<td>8</td>
<td>1 0</td>
</tr>
<tr>
<td>9</td>
<td>1 0</td>
</tr>
</tbody>
</table>

![Graph showing degree distribution with nodes and their in/out-degrees.](image-url)
Degree distribution

Map: for each node, output in- and out-degree with count (of 1)

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 4</td>
</tr>
<tr>
<td>10</td>
<td>0 1</td>
</tr>
<tr>
<td>2</td>
<td>3 2</td>
</tr>
<tr>
<td>3</td>
<td>2 3</td>
</tr>
<tr>
<td>4</td>
<td>2 2</td>
</tr>
<tr>
<td>5</td>
<td>1 0</td>
</tr>
<tr>
<td>6</td>
<td>2 0</td>
</tr>
<tr>
<td>7</td>
<td>0 2</td>
</tr>
<tr>
<td>8</td>
<td>1 0</td>
</tr>
<tr>
<td>9</td>
<td>1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_3</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_1</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_3</td>
<td>1</td>
</tr>
<tr>
<td>out_4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in/out degree count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>0 2</td>
</tr>
<tr>
<td>1 3</td>
</tr>
<tr>
<td>2 4</td>
</tr>
<tr>
<td>3 1</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>0 4</td>
</tr>
<tr>
<td>1 1</td>
</tr>
<tr>
<td>2 3</td>
</tr>
<tr>
<td>3 1</td>
</tr>
<tr>
<td>4 1</td>
</tr>
</tbody>
</table>
Degree distribution

Shuffle: collect counts for each in/out-degree

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-degree</th>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>in_3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>in_0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-degree</th>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>out_1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>out_3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>out_4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_3</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_1</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_3</td>
<td>1</td>
</tr>
<tr>
<td>out_4</td>
<td>1</td>
</tr>
</tbody>
</table>

in/out degree count

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Large-scale social media analysis w/ Hadoop
## Degree distribution

### Reduce: add counts

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 4</td>
</tr>
<tr>
<td>10</td>
<td>0 1</td>
</tr>
<tr>
<td>2</td>
<td>3 2</td>
</tr>
<tr>
<td>3</td>
<td>2 3</td>
</tr>
<tr>
<td>4</td>
<td>2 2</td>
</tr>
<tr>
<td>5</td>
<td>1 0</td>
</tr>
<tr>
<td>6</td>
<td>2 0</td>
</tr>
<tr>
<td>7</td>
<td>0 2</td>
</tr>
<tr>
<td>8</td>
<td>1 0</td>
</tr>
<tr>
<td>9</td>
<td>1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_0</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_1</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_2</td>
<td>1</td>
</tr>
<tr>
<td>in_3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_0</td>
<td>1</td>
</tr>
<tr>
<td>out_1</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_2</td>
<td>1</td>
</tr>
<tr>
<td>out_3</td>
<td>1</td>
</tr>
<tr>
<td>out_4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>0 2</td>
</tr>
<tr>
<td>in</td>
<td>1 3</td>
</tr>
<tr>
<td>in</td>
<td>2 4</td>
</tr>
<tr>
<td>in</td>
<td>3 1</td>
</tr>
<tr>
<td>out</td>
<td>0 4</td>
</tr>
<tr>
<td>out</td>
<td>1 1</td>
</tr>
<tr>
<td>out</td>
<td>2 3</td>
</tr>
<tr>
<td>out</td>
<td>3 1</td>
</tr>
<tr>
<td>out</td>
<td>4 1</td>
</tr>
</tbody>
</table>
Clustering coefficient

Fraction of edges amongst a node’s in/out-neighbors

— e.g. how many of a node’s friends are following each other?
Clustering coefficient

![Graph showing clustering coefficient distribution]

@jakehofman  (Yahoo! Research)  Large-scale social media analysis w/ Hadoop  May 23, 2010  56 / 71
Clustering coefficient
Clustering coefficient
Clustering coefficient
Clustering coefficient

Map: pass all of a node’s out-neighbors to each of its in-neighbors

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 7 3 5 4 6</td>
</tr>
<tr>
<td>2</td>
<td>10 3 4 9 8</td>
</tr>
<tr>
<td>3</td>
<td>1 4 1 2 4</td>
</tr>
<tr>
<td>4</td>
<td>1 3 3 2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1 7</td>
</tr>
<tr>
<td>7</td>
<td>1 6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>out neighbor</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 1</td>
<td>3 5 4 6</td>
</tr>
<tr>
<td>2</td>
<td>9 8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2 9 8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2 9 8</td>
<td></td>
</tr>
</tbody>
</table>

@jakehofman (Yahoo! Research)
### Clustering coefficient

**Shuffle: collect each node’s two-hop neighborhoods**

<table>
<thead>
<tr>
<th>node</th>
<th>in/out-neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 7 3 5 4 6</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10 3 4 9 8</td>
</tr>
<tr>
<td>3</td>
<td>1 4 1 2 4</td>
</tr>
<tr>
<td>4</td>
<td>1 3 3 2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1 7</td>
</tr>
<tr>
<td>7</td>
<td>1 6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>out neighbor</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 1 2 4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4 3 2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2 9 8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1 3 5 4 6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2 9 8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4 3 2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2 9 8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 1 2 4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 3 5 4 6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Clustering coefficient

**Reduce:** count a half-triangle for each node reachable by both a one- and two-hop path

<table>
<thead>
<tr>
<th>node</th>
<th>out neighbor</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1 2 4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3 2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 4</td>
</tr>
<tr>
<td>1</td>
<td>3 2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2 9 8</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1 3 5 4 6</td>
</tr>
<tr>
<td>3</td>
<td>2 9 8</td>
</tr>
<tr>
<td>3</td>
<td>4 3 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2 9 8</td>
</tr>
<tr>
<td>4</td>
<td>3 1 2 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>two-hop neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1 3 5 4 6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
Clustering coefficient

Note: this approach generates large amount of intermediate data relative to final output.
Breadth first search

Iterative approach: each MapReduce round expands the frontier

Map: If node’s distance $d$ to source is finite, output neighbor’s distance as $d+1$
Reduce: Set node’s distance to minimum received from all in-neighbors
Breadth first search

Iterative approach: each MapReduce round expands the frontier

Map: If node’s distance $d$ to source is finite, output neighbor’s distance as $d+1$
Reduce: Set node’s distance to minimum received from all in-neighbors
Breadth first search

Iterative approach: each MapReduce round expands the frontier

Map: If node’s distance $d$ to source is finite, output neighbor’s distance as $d+1$
Reduce: Set node’s distance to minimum received from all in-neighbors
Breadth first search

Iterative approach: each MapReduce round expands the frontier

Map: If node’s distance $d$ to source is finite, output neighbor’s distance as $d+1$
Reduce: Set node’s distance to minimum received from all in-neighbors
Break complicated tasks into multiple, simpler MapReduce rounds.
Pagerank

**Iterative approach:** each MapReduce round broadcasts and collects edge messages for power method $^3$

Map: Output current pagerank over degree to each out-neighbor
Reduce: Sum incoming probabilities to update estimate


$^3$Extra rounds for random jump, dangling nodes
Machine learning
Machine learning

- Often use MapReduce for feature extraction, then fit/optimize locally
- Useful for “embarrassingly parallel” parts of learning, e.g.
  - parameters sweeps for cross-validation
  - independent restarts for local optimization
  - making predictions on independent examples
- Remember: MapReduce isn’t always the answer
Classification

Example: given words in an article, assign article to one of $K$ classes

“Floyd Landis showed up at the Tour of California”
Classification

Example: given words in an article, assign article to one of $K$ classes

“Floyd Landis showed up at the Tour of California”
Classification

Example: given words in an article, assign article to one of $K$ classes

“Floyd Landis showed up at the Tour of California”
Classification: naive Bayes

- Model presence/absence of each word as independent coin flip

\[ p(\text{word}|\text{class}) = \text{Bernoulli}(\theta_{wc}) \]

\[ p(\text{words}|\text{class}) = p(\text{word}_1|\text{class}) p(\text{word}_2|\text{class}) \ldots \]
Classification: naive Bayes

• Model presence/absence of each word as independent coin flip

\[ p(\text{word}|\text{class}) = \text{Bernoulli}(\theta_{wc}) \]
\[ p(\text{words}|\text{class}) = p(\text{word}_1|\text{class}) p(\text{word}_2|\text{class}) \ldots \]

• Maximum likelihood estimates of probabilities from word and class counts

\[ \hat{\theta}_{wc} = \frac{N_{wc}}{N_c} \]
\[ \hat{\theta}_c = \frac{N_c}{N} \]
Classification: naive Bayes

- Model presence/absence of each word as independent coin flip

\[ p(\text{word}|\text{class}) = \text{Bernoulli}(\theta_{wc}) \]
\[ p(\text{words}|\text{class}) = p(\text{word}_1|\text{class}) p(\text{word}_2|\text{class}) \ldots \]

- Maximum likelihood estimates of probabilities from word and class counts

\[ \hat{\theta}_{wc} = \frac{N_{wc}}{N_c} \]
\[ \hat{\theta}_c = \frac{N_c}{N} \]

- Use bayes’ rule to calculate distribution over classes given words

\[ p(\text{class}|\text{words}, \Theta) = \frac{p(\text{words}|\text{class}, \Theta) p(\text{class}, \Theta)}{p(\text{words}, \Theta)} \]
Classification: naive Bayes

Naive $\leftrightarrow$ independent features
Classification: naive Bayes

Naive $\leftrightarrow$ independent features

Class-conditional word count
Classification: naive Bayes

class word count

| sports  | an       | 355 |
| sports  | be       | 317 |
| sports  | first    | 318 |
| sports  | game     | 379 |
| sports  | has      | 374 |
| sports  | have     | 284 |
| sports  | one      | 296 |
| sports  | said     | 325 |
| sports  | season   | 295 |
| sports  | team     | 279 |
| sports  | their    | 334 |
| sports  | this     | 293 |
| sports  | when     | 290 |
| sports  | who      | 363 |
| world   | after    | 347 |
| world   | back     | 299 |
| world   | government | 300 |
| world   | had      | 352 |
| world   | have     | 342 |
| world   | he       | 355 |
| world   | its      | 308 |
| world   | mr       | 293 |
| world   | united   | 313 |
| world   | were     | 319 |

class words

world   Economics Is on Agenda for U.S. Meetings in China
world   U.K. Backs Germany's Effort to Support Euro
sports  A Pitchers' Duel Ends in Favor of the Yankees
sports  After Doping Allegations, a Race for Details
Clustering:

Find clusters of “similar” points

Clustering: K-means

Map: Assign each point to cluster with closest mean, output (cluster, features)

Reduce: Update clusters by calculating new class-conditional means

---

Demonstration of the standard algorithm

1) k initial "means" (in this case k=3) are randomly selected from the data set (shown in color).
2) k clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.
3) The centroid of each of the k clusters becomes the new means.
4) Steps 2 and 3 are repeated until convergence has been reached.

---


---

4 Each mapper loads all cluster centers on init
Mahout

Apache Lucene Mahout

Mahout’s goal is to build scalable machine learning libraries. With scalable we mean:

- Scalable to reasonably large data sets. Our core algorithms for clustering, classification and batch based collaborative filtering are implemented on top of Apache Hadoop using the map/reduce paradigm. However we do not restrict contributions to Hadoop based implementations. Contributions that run on a single node or on a non-Hadoop cluster are welcome as well. The core libraries are highly optimized to allow for good performance also for non-distributed algorithms.
- Scalable to support your business case. Mahout is distributed under a commercially friendly Apache Software License.
- Scalable community. The goal of Mahout is to build a vibrant, responsive, diverse community to facilitate discussions not only on the project itself but also on potential use cases. Come to the mailing lists to find out more.

Currently Mahout supports mainly four use cases: Recommendation mining takes users’ behavior and from that tries to find items users might like. Clustering takes e.g. text documents and groups them into groups of topically related documents. Classification learns from existing categorized documents what documents of a specific category look like and is able to assign unlabelled documents to the (hopefully) correct category. Frequent itemset mining takes a set of item groups (terms in a query session, shopping cart content) and identifies, which individual items usually appear together.

http://mahout.apache.org/
Thanks

- Sharad Goel\textsuperscript{y}
- Winter Mason\textsuperscript{y}
- Sid Suri\textsuperscript{y}
- Sergei Vassilvitskii\textsuperscript{y}
- Duncan Watts\textsuperscript{y}
- Eytan Bakshy\textsuperscript{m, y}

\textsuperscript{y} Yahoo! Research (http://research.yahoo.com)
\textsuperscript{m} University of Michigan
Thanks.

Questions?[^5]