CSE 544: Principles of Database Systems

MapReduce, PigLatin

Overview of Today's Lecture

- Cluster computing
  - Map/reduce (paper)
- Degree sequence (brief discussion)
- PigLatin (brief discussion, time permitting)

Next lecture (Monday):

- Parallel data processing wrapup, including query processing from lecture 9
- Start discussing datalog

Cluster Computing

Cluster Computing

- Large number of commodity servers, connected by high speed, commodity network
- Rack: holds a small number of servers
- Data center: holds many racks

Cluster Computing

- Massive parallelism:
  - 100s, or 1000s, or 10000s servers
  - Many hours
- Failure becomes a fact of life:
  - If medium-time-between-failure is 1 year
  - Then 10000 servers have one failure / hour

Distributed File System (DFS)

Very large files: TBs, PBs

- 1 file = partitioned into chunks, e.g. 64MB
- 1 chunk = replicated several times (≥3)

Implementations:

- Google's DFS: GFS, proprietary
- Hadoop's DFS: HDFS, open source

MapReduce

- Google: paper published 2004
- Free variant: Hadoop

- MapReduce = high-level programming model and implementation for large-scale parallel data processing

Data Model

Files !

A file = **`BagOf(key, value)`**

A MapReduce Program:

- Input: **`BagOf(input_key,input_value)`**
- Output: **`BagOf(output_key, output_value)`**

The Map and Reduce Functions

User provides the MAP-function:

- Input: **`(input_key, input_value)`**
- Ouput: **`BagOf(intermediate_key, intermediate_value)`**

User provides the REDUCE function:

- Input: **`(intermediate_key, BagOf(intermediate_values))`**
- Output: **`BagOf(output_values)`**

Doc(key, word) → denormalized to Doc(key, value) where value = BagOf(word)

Example

- Counting the number of occurrences of each word in a large collection of docs
- Each document
  - The key = document id
  - The value = set of words

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

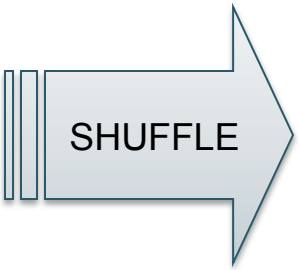
```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Doc(key, word)

MAP

REDUCE

Input	Output
(key1,'to be or not to be…')	(to,1)
	(be,1)
	(or,1)
	(not,1)
	(to,1)
	(be,1)
(key2,'to buy or not to buy…')	(to,1)
	(buy,1)
	…



Input	Output
(be,[1,1,…])	(be,705)
(buy,[1,1,…])	(buy,666)
(or,[1,1,1,…])	(or,1032)
(to,[1,1,1,…])	(to,….)
…	

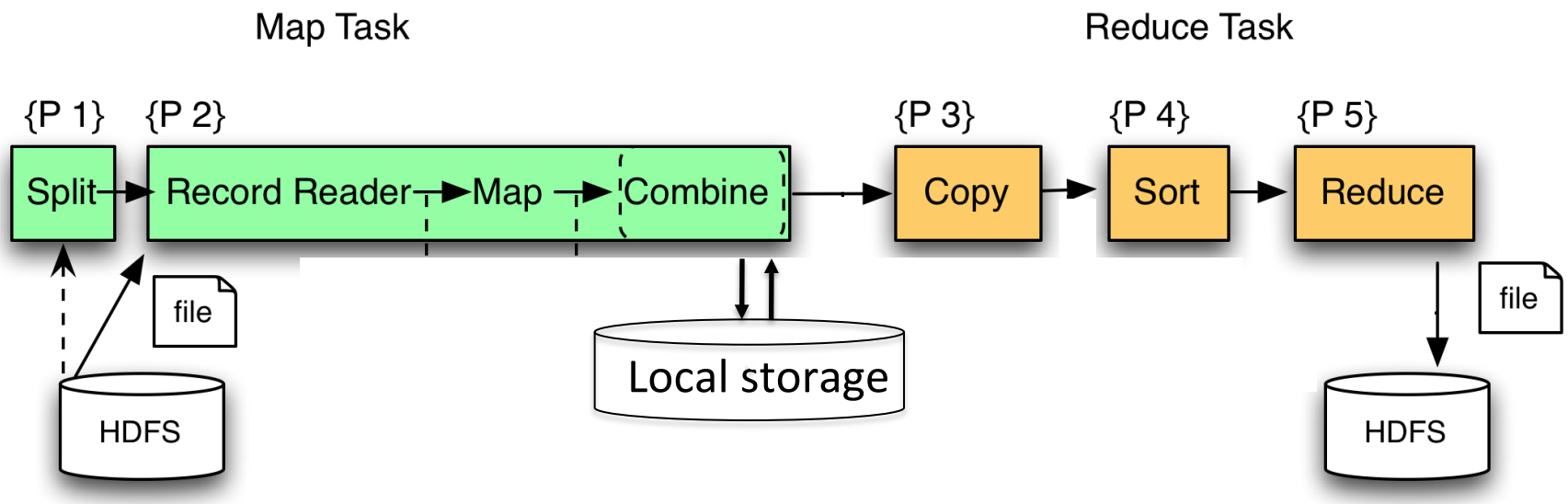
Doc(key, word)

MAP=GROUP BY, REDUCE=Aggregate

```
SELECT word, sum(1)
FROM Doc
GROUP BY word
```

MR Phases

- Each Map and Reduce task has multiple phases:



Implementation

- One master node
- Input file partitioned into *M splits* by key
- Master assigns the *M map tasks* to *workers*
- Workers write output to local disk: *R regions*
- Master assigns *R reduce tasks* to *workers*
- Reduce workers *copy* regions from the map workers' local disks

Failures, Stragglers

Worker failure

- Master pings workers periodically,
- If down then reassigns the task to another worker

Straggler

- A machine that takes unusually long time to complete one of the last tasks.

Tuning MapReduce

Very hard!

- Choosing M,R:
  - Bigger is better but master needs O(M×R) memory
- Typical:
  - M=number of chunks (“number of blocks”)
  - R=much smaller (why??); e.g. 1.5 * #servers
- The combiner (Talk in class…)

MapReduce Summary

- Hides scheduling and parallelization details
- However, very limited queries
  - Difficult to write more complex tasks
  - Need multiple map-reduce operations
- Solution: declarative query language over MR: PigLatin, Dryad, Dremmel, Tenzing, …

Degree Sequence

Famous Example of Big Data Analysis

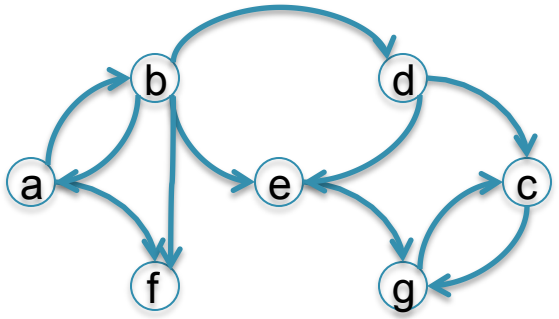
Kumar et al., *The Web as a Graph*

- Question 1: is the Web like a "random graph"?
- Question 2: how does the Web graph look like?

Graph as Databases

Many large databases are graphs

- The Web
- The Internet
- Social Networks
- Flights btw. Airports
- Etc,etc,etc



Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

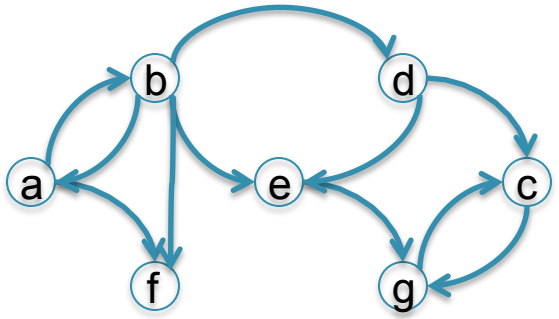
Data Analytics on Big Graphs

Queries expressible in SQL:

- How many nodes (edges)?
- How many nodes have > 4 neighbors?
- Which are the "most connected nodes"?

Queries requiring recursion:

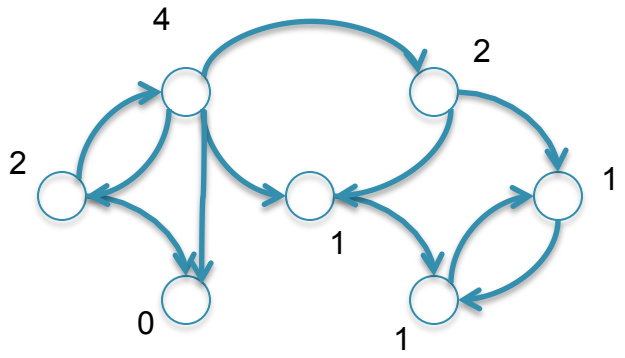
- Is the graph connected?
- What is the diameter of the graph?
- Compute *PageRank*
- Compute the *Centrality* of each node



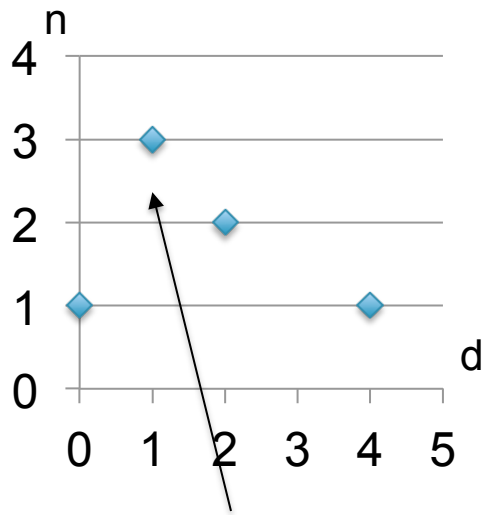
Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Histogram of a Graph a.k.a. Degree Sequence

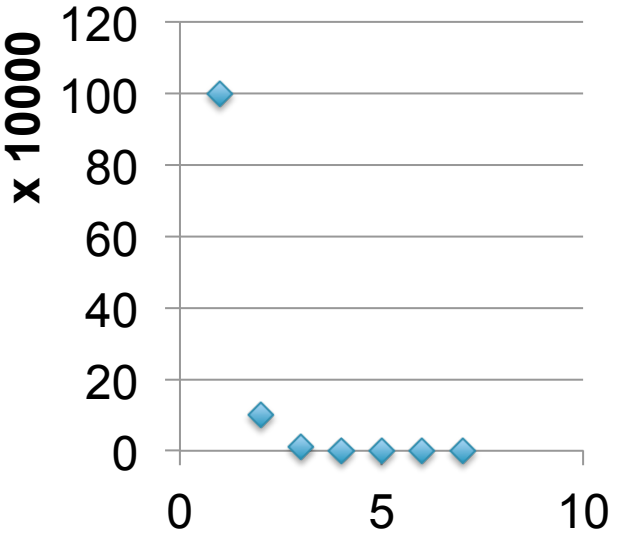
- Outdegree of a node = number of outgoing edges
- For each d, let n(d) = number of nodes with oudegree d
- The outdegree histogram of a graph = the scatterplot (d, n(d))

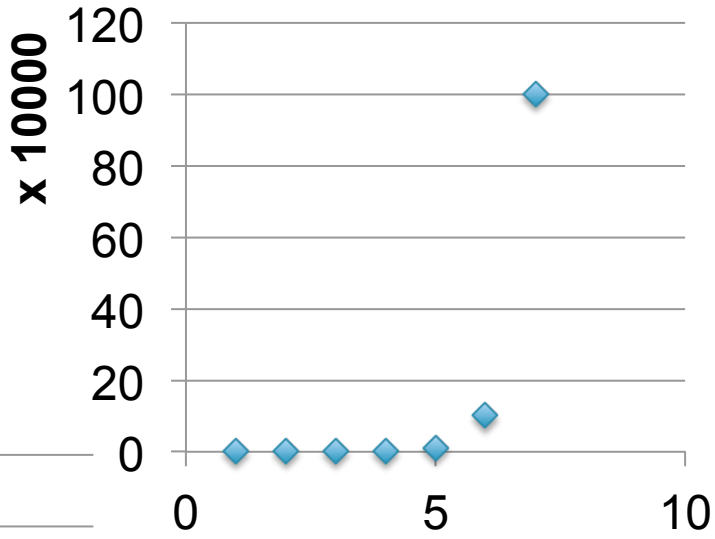


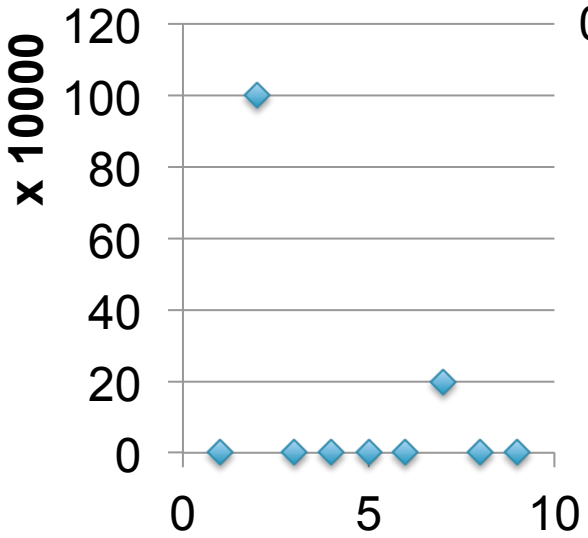
d	n(d)
0	1
1	3
2	2
3	0
4	1



Histograms Tell Us Something About the Graph



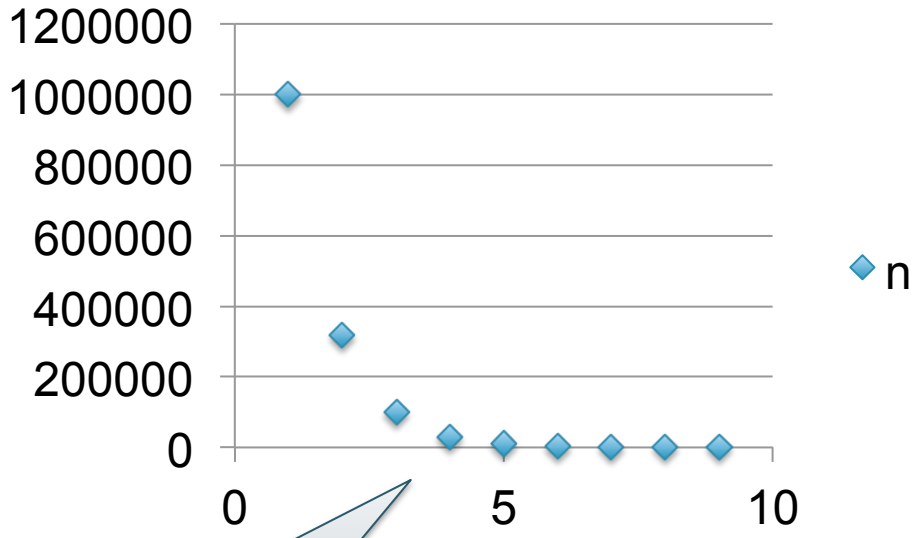


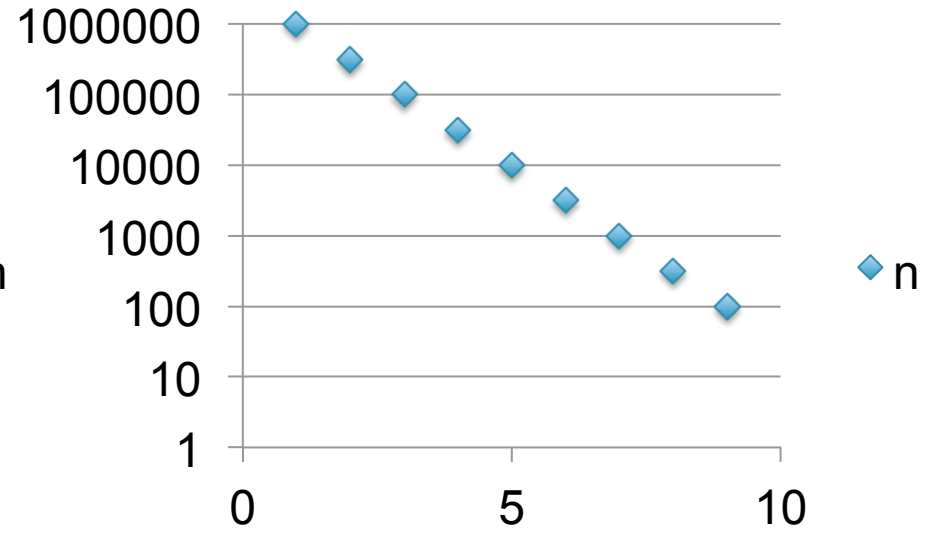


What can you say about these graphs?

Exponential Distribution

- $n(d) \cong c/2^d$ (generally, cx^d, for some $x < 1$)
- A *random graph* has exponential distribution
- Best seen when n is on a log scale

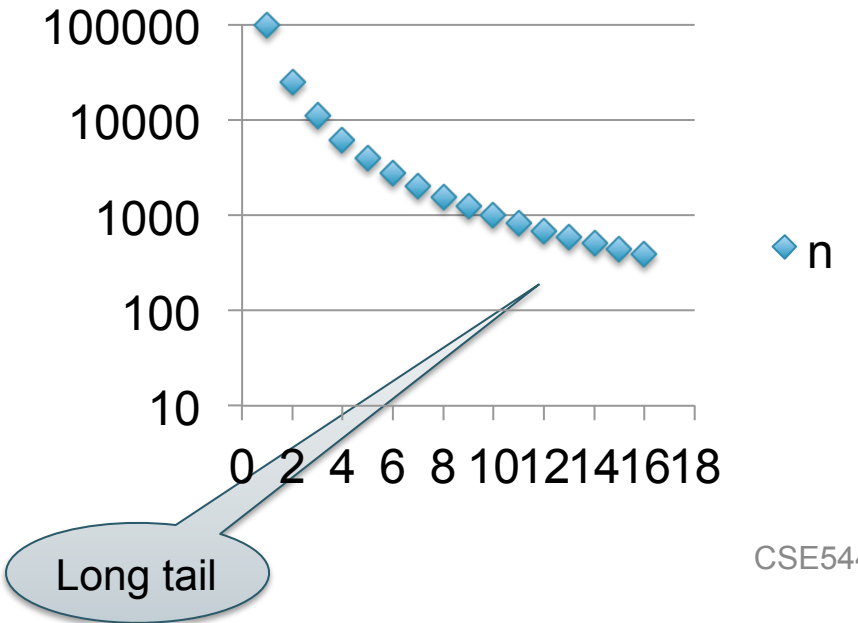


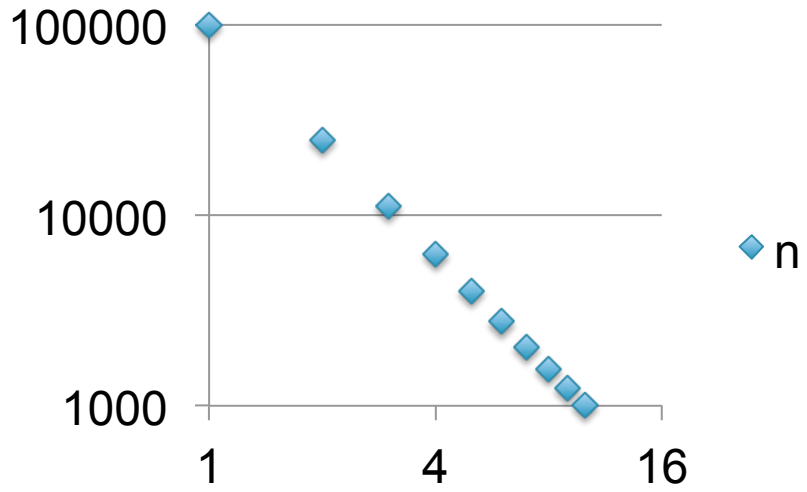


Quickly vanishing

Zipf Distribution

- $n(d) \cong 1/d^x$, for some value $x>0$
- Human-generated data has Zipf distribution: letters in alphabet, words in vocabulary, etc.
- Best seen in a log-log scale





The Histogram of the Web

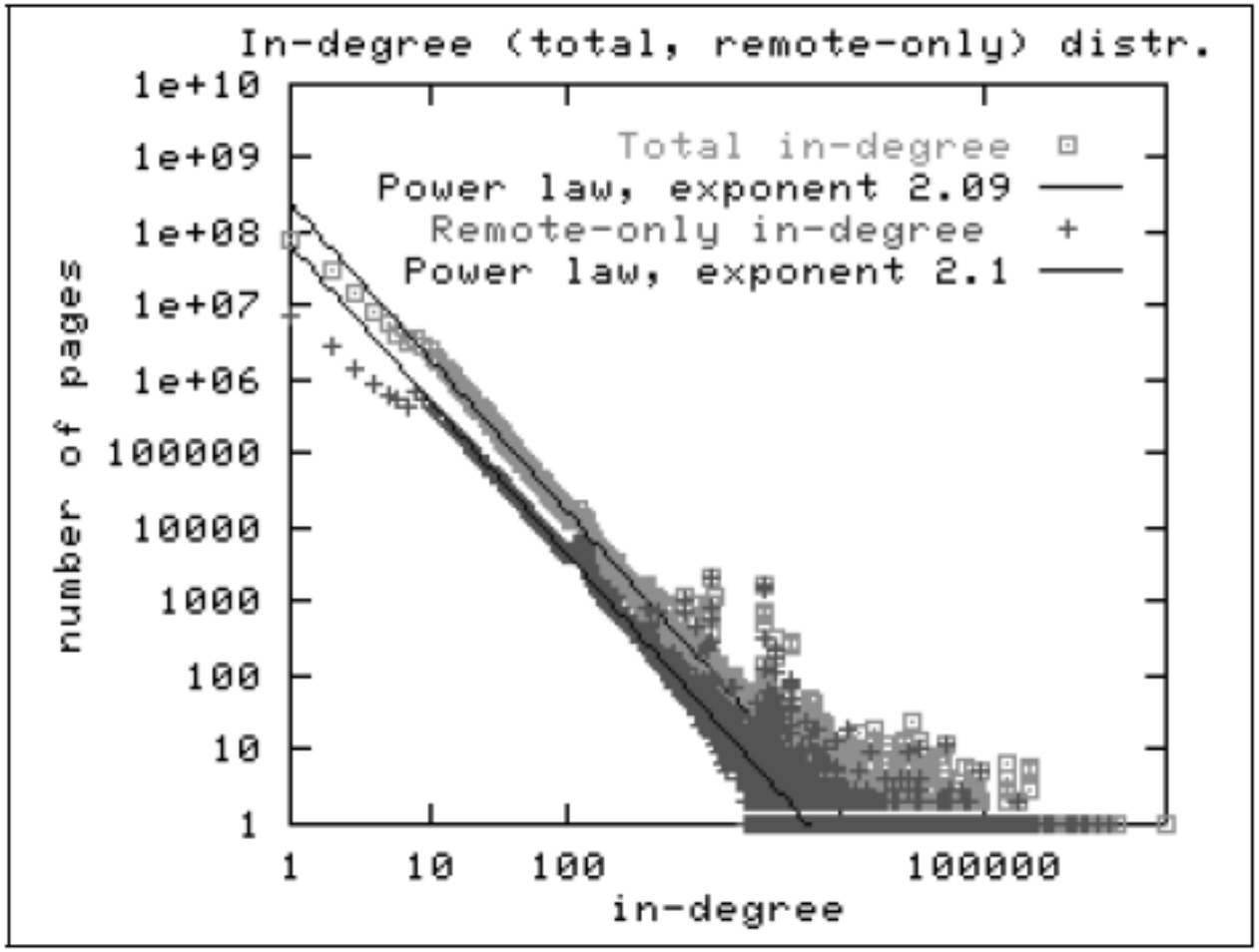


Figure 2: In-degree distribution.

Late 1990's
200M Webpages

Exponential ?

Zipf ?

The Bowtie Structure of the Web

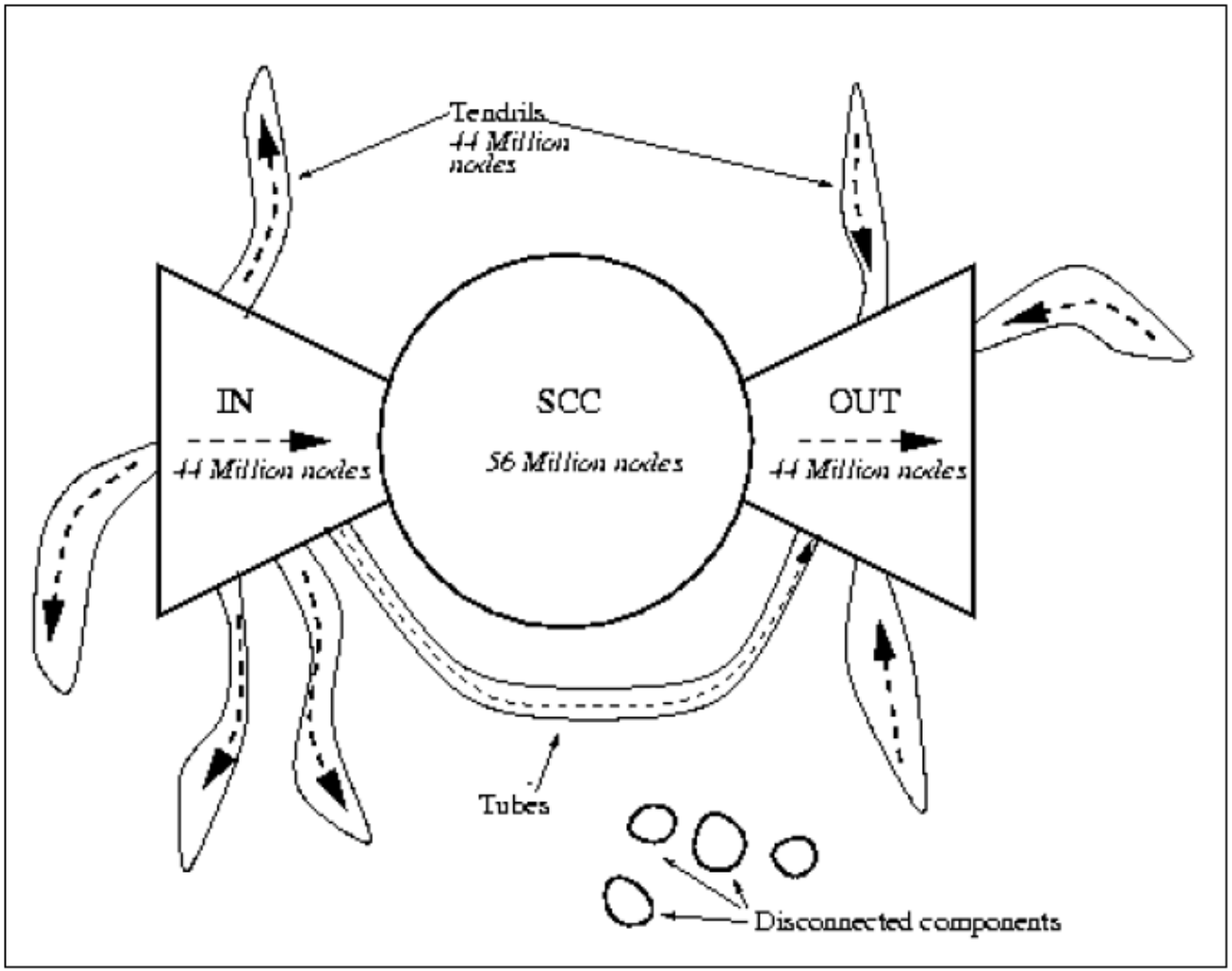


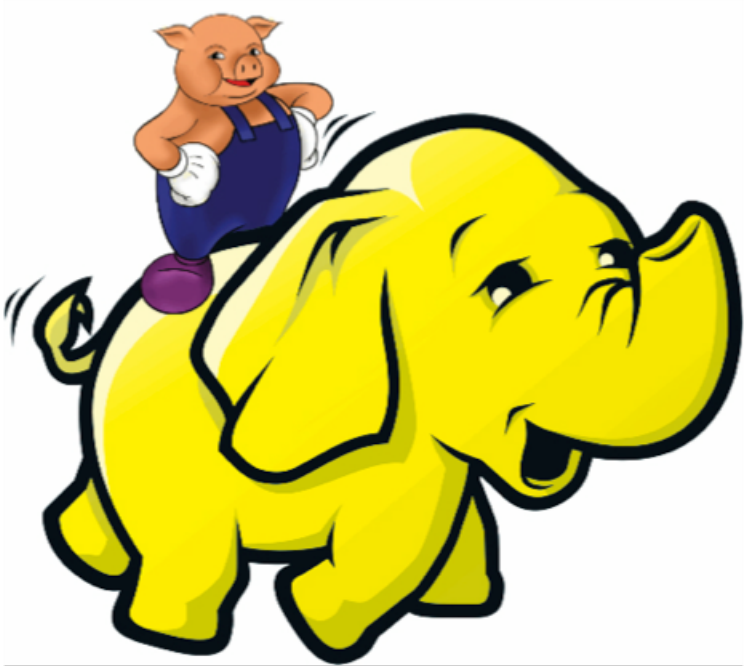
Figure 4: The web as a bowtie. SCC is a giant strongly connected component. IN consists of pages with paths to SCC, but no path from SCC. OUT consists of pages with paths from SCC, but no path to SCC. TENDRILS consists of pages that cannot surf to SCC, and which cannot be reached by surfing from SCC.

Pig Latin

Slides courtesy of: Alan Gates, Yahoo!Research

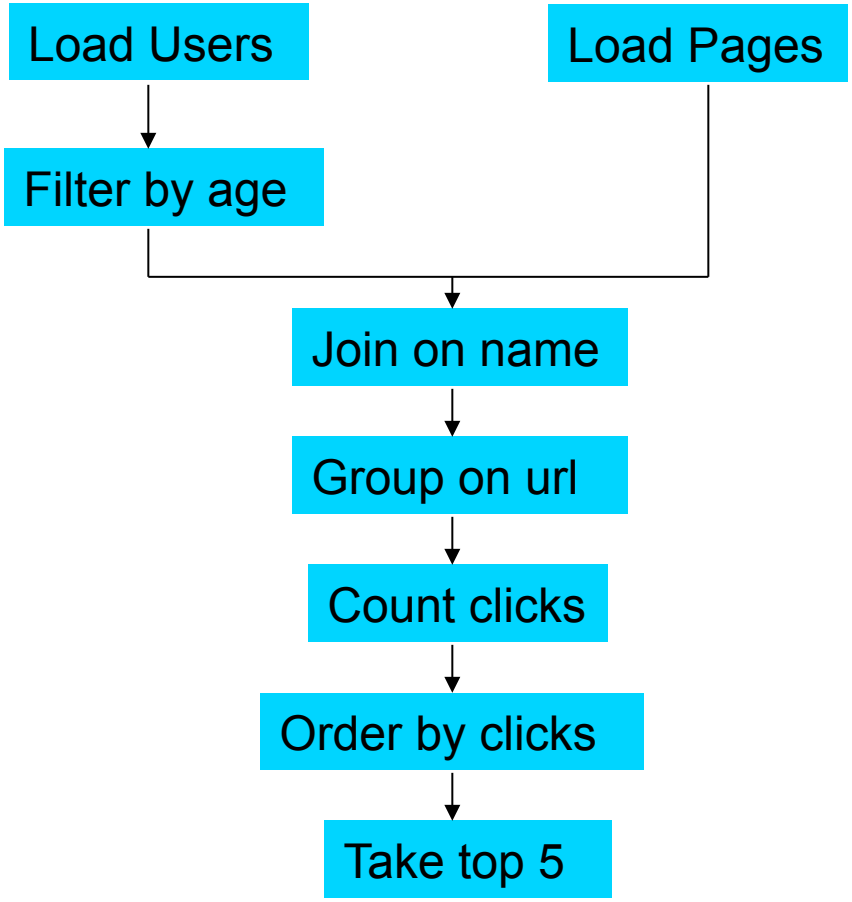
What is Pig?

- An engine for executing programs on top of Hadoop
- It provides a language, Pig Latin, to specify these programs
- An Apache open source project
  http://hadoop.apache.org/pig/



Why use Pig?

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited sites by users aged 18 - 25.



In Map-Reduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
                new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
                new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

       JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

170 lines of code, 4 hours to write

In Pig Latin

```
Users = load ‘users’ as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load ‘pages’ as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into ‘top5sites’;
```

9 lines of code, 15 minutes to write