FAQs

- Term project deliverable 0
  - Item 1: Your team members
  - Item 2: Tentative project titles (up to 3)
  - Submission deadline: Feb. 1
  - Via email or canvas

- PA1
  - Hadoop and Spark installation guides are posted
  - If you would like to start your homework, please send me an email with your team information. I will assign the port range for your team.

- Quiz 1: February 4, 2019 in class

Topics of Today’s Class

- Overview of Programming Assignment 1
- Distributed Computing Models for Scalable Batch Computing
- MapReduce

Programming Assignment 1
Hyperlink-Induced Topic Search (HITS)

This material is built based on


Types of Web queries

- Yes/No queries
  - Does Chrome support .ogg video format?

- Broad topic queries
  - Find information about “polar vortex”

- Similar-page query
  - Find pages similar to https://stackoverflow.com/

http://www.cs.colostate.edu/~cs535
Ranking algorithm to find the most "authoritative" pages

- To find the small set of the most authoritative pages that are relevant to the query

- Examples of the authoritative pages
  - For the topic, "python"
    - https://www.python.org
  - For the information about "Colorado University"
    - https://www.colostate.edu
  - For the images about "iPhone"
    - https://www.apple.com/iphone

Challenge of content-based ranking

- Most useful pages do not include the keyword (that the users are looking for)
  - "computer" in the APPLE page?

- Pages are not sufficiently descriptive
  - "health care" in Poudre Valley Hospital?

HITS (Hypertext-Induced Topic Search)

- Authority
  - A Web page with good, authoritative content on a specific topic
  - A Web page that is linked by many hubs

- Hub
  - A Web page pointing to many authoritative Web pages
  - e.g. portal pages (Yahoo)

- PageRank captures simplistic view of a network

A.K.A. Hubs and Authorities
- Jon Kleinberg 1997
- Topic search
- Automatically determine hubs/authorities

In practice
- Performed only on the result set (PageRank is applied on the complete set of documents)
- Developed for the IBM Clever project
- Used by Teoma (later Ask.com)
Understanding Authorities and Hubs

- Intuitive idea to find authoritative results using link analysis:
  - Not all hyperlinks are related to the conferment of authority
- Patterns that authoritative pages have
  - Authoritative Pages share considerable overlap in the sets of pages that point to them.
- Understanding Authorities and Hubs
  - Intuitive idea to find authoritative results using link analysis:
  - Not all hyperlinks are related to the conferment of authority
  - Authoritative Pages share considerable overlap in the sets of pages that point to them.

Calculating Authority/Hub scores

Let there be $n$ Web pages. Define the $n \times n$ adjacency matrix $A$ such that, $A_{ij} = 1$ if there is a link from $i$ to $j$. Otherwise $A_{ij} = 0$. Each Web page has an authority score $a_i$ and a hub score $h_j$. We define the authority score by summing up the hub scores that point to it, $a_i = \sum_{j=1}^{n} h_j A_{ij}$. This can be written concisely as, $a = Ah$.

Hubs and Authorities

Let's start arbitrarily from $a_0 = \mathbf{1}$, $h_0 = \mathbf{1}$, where $\mathbf{1}$ is the all-one vector.

Let $a_0 = (1, 1, 1)$, $h_0 = (1, 1, 1)$. Repeating this, the sequences $a_0, a_1, a_2, \ldots$, and $h_0, h_1, h_2, \ldots$ converge to limits $a^*$ and $h^*$.


We define the hub score by summing up the row scores of the matrix. $h_j = \sum_{i=1}^{n} a_i A_{ij}$. This can be written concisely as, $h = A^T a$.

Hubs and Authorities

Let’s start arbitrarily from $a_0 = 1$, $h_0 = 1$, where $1$ is the all-one vector.

$a_0 = (1, 1, 1, 1)$
$h_0 = (1, 1, 1, 1)$

$a_1 = \left(\frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\right)$
$h_1 = \left(\frac{1}{8}x_0 + \frac{1}{8}x_1 + \frac{1}{4}x_1 + \frac{1}{2}x_1\right)$

After the normalization:

$h_1 = (7/22, 6/22, 5/22, 4/22)$ (hub values after the first iteration)

Step 1. Constructing a focused subgraph (root set)

- Generate a root set from a text-based search engine
- e.g. pages containing query words

Step 2. Constructing a focused subgraph (base set)

- For each page $p \in R$
  - Add the set of all pages $p$ points to
  - Add the set of all pages pointing to $p$

Step 3. Initial values

<table>
<thead>
<tr>
<th>Pages</th>
<th>Hubs</th>
<th>Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Ranks
Hub: P1>P2>P3>P4
Authority: P1>P2>P3>P4

Step 4. After the first iteration

<table>
<thead>
<tr>
<th>Pages</th>
<th>Hubs</th>
<th>Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>7/22</td>
<td>1/8</td>
</tr>
<tr>
<td>P2</td>
<td>6/22</td>
<td>1/8</td>
</tr>
<tr>
<td>P3</td>
<td>5/22</td>
<td>3/8</td>
</tr>
<tr>
<td>P4</td>
<td>4/22</td>
<td>4/8</td>
</tr>
</tbody>
</table>

Normalization
- Original paper: using squares sum (to 1)
- You can use sum (to 1)
  - value = value/(sum of all values)
Step N. Convergence of scores

- Repeat the calculation (step 4) until the scores converge
- You should specify your threshold (maximum number of N)

Do we need to perform the matrix multiplication?

- Yes/No
- It will be a valid answer
- However, you can consider the random walk style implementation
- Please see examples of PageRank algorithm provided by Apache Spark:
  - https://spark.apache.org/docs/1.6.1/api/java/org/apache/spark/graphx/lib/PageRank.html

3. Distributed Computing Models for Scalable Batch Computing
   Part 1. MapReduce

This material is developed based on,

  - Download this chapter from the CS435 schedule page
- MapReduce Design Patterns, Donald Miner and Adam Shook, O’Reilly, 2013

What is MapReduce?
MapReduce

- **MapReduce** is inspired by the concepts of map and reduce in Lisp.
- “Modern” MapReduce
  - Developed within Google as a mechanism for processing large amounts of raw data.
  - Crawled documents or web request logs
  - Distributes these data across thousands of machines
  - Same computations are performed on each CPU with different dataset

Mapper

- **Mapper** maps input key/value pairs to a set of intermediate key/value pairs
  - Maps are the individual tasks that transform input records into intermediate records
  - The transformed intermediate records do not need to be of the same type as the input records
  - A given input pair may map to zero or many output pairs
  - The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job

Reducer

- **Reducer** reduces a set of intermediate values which share a key to a smaller set of values
  - Reducer has 3 primary phases
    - Shuffle, sort and reduce
  - **Shuffle**
    - Input to the reducer is the sorted output of the mappers
    - The framework fetches the relevant partition of the output of all the mappers via HTTP
  - **Sort**
    - The framework groups input to the reducer by keys

Example 1: NCDC data example

- A national climate data center record
- Find the maximum temperature of a year (1900 ~ 1999)

```
SST
332130 # WSRV weather station identifier
99999 # WSNZ weather station identifier
1990101 # observation date
1110 # observation time
s1 # latitude (degrees x 100)
99999 # longitude (degrees x 100)
FR-12
+ +171 # elevation (meters)
0125 # quality code
```

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University
Analyzing the data with Unix Tools (1/2)

A program for finding the maximum recorded temperature by year from NCDC weather records

```bash
#!/usr/bin/env bash
for year in all/*
do
  echo -ne "basename $year .gz"
  gunzip -c $year |
  awk '{
    temp = substr($0, 88, 5) + 0;
    q = substr($0, 93, 1);
    if (temp != 9999 && q ~ /[01459]/ && temp > max)
      max = temp }
END { print max }'
done
```

Analyzing the data with Unix Tools (2/2)

- The script loops through the compressed year files
- Printing the year
- Processing each file using `awk`
- Extracts two fields: Air temperature and the quality code
- Check if it is greater than the maximum value seen so far

```
./max_temperature.sh
1901 317
1902 244
1903 289
1904 256
1905 283
...```

Results?

- The complete run for the century took 42 minutes
- To speed up the processing
  - We need to run parts of the program in parallel
  - Process different years in different processes
- What will be the problems?

Challenges

- Dividing the work into equal-size pieces
- Data size per year?
- Combining the results from independent processes
  - Combining results and sorting by year?
- You are still limited by the processing capacity of a single machine (the worst one!)

Map and Reduce

- MapReduce works by breaking the processing into two phases
  - The map phase
  - The reduce phase
- Each phase has key-value pairs as input and output
- Programmers should specify
  - Types of input/output key-values
  - The map function
  - The reduce function
Sample lines of input data

These lines are presented to the map function as the key-value pairs

The keys are the line offsets within the file (optional)

The map function extracts the year and the air temperature and emit them as its output

This output key-value pairs will be sorted (by key) and grouped by key
Values passed to each reducer are NOT sorted
Our reduce function will see the following input:

Example 2: WordCount

- For text files stored under /usr/joe/wordcount/input, count the number of occurrences of each word
- How do files and directory look?

Example 2: WordCount

- Run the MapReduce application
Example 2: WordCount

Mappers
- Read a line
- Tokenize the string
- Pass the <key, value> output to the reducer

Reducers
- Collect <key, value> pairs sharing the same key
- Aggregate total number of occurrences

What do you have to pass from the Mappers?

Example 2: WordCount

```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

Example 2: WordCount

```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

Exercise

Design your map and reduce function to perform following data processing.

Find the 10 clients who spent the most electricity (kilowatt) for each zip code for the last month.

Files contain information about the last month only. The data is formatted as follows:

```
{ customerID, TAB, address, TAB, zipcode, TAB, electricity usage, LINEFEED }
```

Assume that each line will be used as the input to a Map function.

Question 1: What are the input/output/functionality of your mapper?

Question 2: What are the input/output/functionality of your reducer?

Comparison with other systems

- MPI vs. MapReduce
  - MapReduce tries to collocate the data with the compute node
  - Data access is fast
  - Data is local

- Volunteer computing vs. MapReduce
  - SETI@home
    - Using donated CPU time
    - What are the differences between MapReduce vs. SETI@home?

MapReduce Data Flow
MapReduce data flow with a single reducer

MapReduce data flow with multiple reducers

Data locality optimization
- Hadoop tries to run the map task on a node where the input data resides in HDFS
  - Minimizes usage of cluster bandwidth
- If all replication nodes are running other map tasks
  - The job scheduler will look for a free map slot on a node in the same rack

Data movement in Map tasks

Shuffle
- The process by which the system performs the sort and transfers the map outputs to the reducers as inputs
  - MapReduce guarantees that the input to every reducer is sorted by key
Combiner functions

- Minimize data transferred between map and reduce tasks
- Users can specify a **combiner function**
  - To be run on the map output
  - To replace the map output with the combiner output

Combiner example

- Example (from the previous max temperature example)
  - The first map produced,
    - (1950, 5), (1950, 20), (1950, 10)
  - The second map produced,
    - (1950, 25), (1950, 15)
  - The reduce function is called with a list of all the values.
    - (1950, [0, 20, 25, 15])
  - Output will be,
    - (1950, 25)
  - We may express the function as,
    - \[ \text{max}(0, 20, 15) \]
      - \[ \text{max}(0, 20, 10, \text{max}(25, 15)) \]
      - \[ \text{max}(20, 25) = 25 \]

Combiner function

- Run a **local** reducer over Map output
- Reduce the amount of data shuffled between the mappers and the reducers
- Combiner cannot replace the reduce function
- Why?

Questions?