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 the Programming Assignment 1
- 3. 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/
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 State University"
    - https://www.colostate.edu

  - For the images about "iPhone"
    - https://www.apple.com/iphone

Challenge of content-based ranking

- How about IBM’s web page?

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

HITS (Hipertext-Induced Topic Search)

- PageRank captures simplistic view of a network

  - 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)

- 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)
Calculating Authority/Hub scores [1/3]

Let there be \( n \) Web pages

Define the \( n \times n \) adjacency matrix \( A \) such that,

\[
A_{ij} = \begin{cases} 
1 & \text{if there is a link from } i \text{ to } j \\
0 & \text{otherwise}
\end{cases}
\]

Otherwise \( A_{ii} = 0 \)

Graph with pages

\[
\begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]

This can be written concisely as,

\[
h = Aa
\]

Calculating Authority/Hub scores [2/3]

Each Web page has an authority score \( a \) and a hub score \( h \).

We define the authority score by summing up the hub scores that point to it,

\[
a_i = \sum_{j=1}^{n} h_j a_{ij}
\]

Graph with pages

\[
\begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

This can be written concisely as,

\[
a = Ah
\]

Understanding Authorities and Hubs [1/2]

Intuitive idea to find authoritative results using link analysis:

- Not all hyperlinks are related to the conferment of authority

- Patterns that authoritative pages have

\[ \text{Authorities} \]

\[ \text{Hubs} \]

Similarly, we define the hub score by summing up the row # in the matrix

\[
h_i = \sum_{j=1}^{n} a_j a_{ij}
\]

Graph with pages

\[
\begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Understanding Authorities and Hubs [2/2]

- A good hub page points to many good authoritative pages

- A good authoritative page is pointed to by many good hub pages

- Authorities and hubs have a mutual reinforcement relationship

Hubs and Authorities

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

\[
a_0 = (1, 1, 1, 1),\]

\[
h_0 = (1, 1, 1, 1)
\]

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

\[
a^* = (1/4, 1/4, 1/4, 1/4),
\]

\[
h^* = (1/4, 1/4, 1/4, 1/4)
\]

Normalizing it: \( (1/4, 1/4, 1/4, 1/4), (1/4, 1/4, 1/4, 1/4) = (1, 1, 1, 1) \)

\[
4(1/4 + 1/4 + 1/4 + 1/4) = (1/8, 1/8, 1/8, 1/8)
\]

\[
a = (1/8, 1/8, 1/8, 1/8) \ (\phi = \text{authority values after the first iteration})
\]

http://www.cs.colostate.edu/~cs535
Hubs and Authorities

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

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

After the normalization:

\[
\begin{bmatrix}
7/22 & 6/22 & 5/22 & 4/22 \\
\end{bmatrix}
\]

(β hub values after the first iteration)

Implementing Topic Search using HITS

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>Node</th>
<th>Hub</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>Node</th>
<th>Hub</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>1/8</td>
</tr>
<tr>
<td>P4</td>
<td>4/22</td>
<td>1/8</td>
</tr>
</tbody>
</table>

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

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

http://www.cs.colostate.edu/~cs535
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](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
   a. Introduction to 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?

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

```
0057 332130 # USAF weather station identifier
9999 # WBAN weather station identifier
19500101 # observation date
0300 # observation time 4
+ 51317 # latitude (degrees x 1000)
+ 028783 # longitude (degrees x 1000)
FM-12
+ 0171 # elevation (meters)
9999
V020 320 # wind direction (degrees)
1 # quality code
```

http://www.cs.colostate.edu/~cs535
The first entries for 1990

```
ls raw/1990 | head -n 10
```

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 -n `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
Visualizing the way the MapReduce works (1/3)

Sample lines of input data:

0840109099999510015107204...99999999 + 06801 + 99999999999...
0840109099999510015101208...99999999999999 + 06802 + 99999999999...
0840109099999510015111104...99999999999999 + 06803 + 99999999999...
0840109099999510015121604...99999999999999 + 06804 + 99999999999...

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

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

Visualizing the way the MapReduce works (2/3)

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

- $(1949, 0)$
- $(1950, 22)$
- $(1950, -11)$
- $(1949, 111)$
- $(1949, 78)$

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:

- $(1949, [111, 78])$
- $(1950, [0, 22, -11])$

Reduce function iterates through the list and pick up the maximum reading. This is the final output.

Visualizing the way the MapReduce works (3/3)

Reduce function iterates through the list and pick up the maximum reading.

Input

1949, 111
1950, 22
1949, 111
1950, 22
1949, 111
1950, 22
1949, 111
1950, 22

This is the final output.

MapReduce Example 2

Example 2: WordCount (1/5)

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

```bash
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World, bye World!
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop, Goodbye to hadoop.
```

Example 2: WordCount (2/5)

- Run the MapReduce application:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.apache.hadoop.mapreduce.WordCount
/usr/joe/wordcount/input /usr/joe/wordcount/output
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
bye 1
Hadoop 1
World 1
Hello 2
bye 1
```

```bash
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
bye 1
Hadoop 1
World 1
Hello 2
bye 1
```
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, zip code, 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?

Answer

- Assume that all the ClientIDs are unique.

1. Mapper
   Input: <dummy key (e.g. file offset number), a line of the input file (e.g. customerID, TAB, address, TAB, zip code, TAB, electricity usage, LINEFEED)>
   Functionality:
   Tokenize the string and retrieve the zip code
   Generate an output
   Output: <zip code, [customer ID, electricity usage]>

2. Reducer
   Input: <zip code, a list of [customer ID, electricity usage]>
   Functionality:
   Scan the list of values and identify top 10 customers with highest electricity usages
   Output: <zip code, a list of customers>

Better Answer: Top-N design pattern

- Assume that all the ClientIDs are unique.

1. Mapper
   Input: <dummy key (e.g. file offset number), a line of the input file (e.g. customerID, TAB, address, TAB, zip code, TAB, electricity usage, LINEFEED)>
   Functionality:
   Create a data structure (HashMap: local_top10) to store the local top 10 information
   Tokenize the string and retrieve the zip code
   If this client is considered as one of the local top 10 until this point, update local_top10
   Add the input split is completely scanned, generate output with local_top10
   Output: [zip code, local_top10]

2. Reducer
   Input: <zip code, a list of [local_top10]>
   Functionality:
   Scan the list of values and identify top 10 customers with highest electricity usages
   Output: <zip code, a list of customers>

   This approach will reduce the communication within your MR cluster significantly.
Better Answer: Top-N design pattern: More Info

Structure of the Top-N pattern

Input → Split → Filter → Mapper → Local Top 10 → Top Ten Reducer → Top Ten output

Input → Split → Filter → Mapper → Local Top 10 → Top Ten Reducer → Top Ten output

public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
  private TreeMap<Integer, Text> LocalTop10 = new TreeMap<Integer, Text>;

  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    // Your code to extract zip code and other attributes, if there are multiple zip code you can
    // retrieve corresponding TopTen based on the zip code here.
    // Your code to evaluate current electricity usage: Add this value and remove the lowest value
    LocalTop10.put(Integer.parseInt(electricity_usage), new Text(your_value));
    if (repToRecordMap.size() > 10) {
      repToRecordMap.remove(repToRecordMap.firstKey());
    }
  }

  protected void cleanup(Context context) throws IOException, InterruptedException {
    for (Text t : repToRecordMap.values()) {
      context.write(zipcode, t);
    }
  }
}

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 with a single reducer

http://www.cs.colostate.edu/~cs535
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 an available map slot on a node in the same rack

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

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

- Example (from the previous max temperature example)
  - The first map produced:
    - (1950, 0), (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, 10, 25, 15])
  - Output will be:
    - (1950, 25)
- We may express the function as:
  - \( \max(0, 20, 10, 25, 15) = \max(\max(0, 20, 10), \max(25, 15)) = \max(20, 25) = 25 \)

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?