PART A. BIG DATA TECHNOLOGY
3. DISTRIBUTED COMPUTING MODELS FOR SCALABLE BATCH COMPUTING

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- TP0
  - There may be adjustment of your team composition

- PA1
  - Hadoop and Spark installation video clips are posted

http://www.cs.colostate.edu/~cs535
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 .ogv video format?

- **Broad topic queries**
  - Find information about "Coronavirus"

- **Similarity query**
  - Find person similar to "Justin Bieber"

---

Challenge of content-based ranking for topic search

- Assume that you are looking for "computer"
  - "computer" in the APPLE page?

- How about IBM’s web page?
Challenge of content-based ranking for topic search

• Most useful pages do not include the keyword (that the users are looking for)

• Pages are not sufficiently descriptive!
  • Semantic mismatch
  • Search keys vs. descriptions
Ranking algorithm to find the most “authoritative” pages for the given topic

- 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/

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)
HITS (Hypertext-Induced Topic Search)

- 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 conferral 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

- 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**

Calculating Authority/Hub scores

Let there be $n$ Web pages
Define the $n \times n$ adjacency matrix $A$ such that,

$$A_{uv} = \begin{cases} 1 & \text{if there is a link from } u \text{ to } v. \\ 0 & \text{otherwise} \end{cases}$$

Graph with pages

$$
\begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$
Calculating Authority/Hub scores

Each Web page $i$ has an authority score $a_i$ and a hub score $h_i$.
We define the authority score by summing up the hub scores that point to it,

$$a_i = \sum_{j=1}^{n} h_j A_{ji}$$

$j$: row # in the matrix
$i$: column # in the matrix

This can be written concisely as,

$$a = A^t h$$

Similarly, we define the hub score of a Web page $i$ by summing up the authority scores $a_j$,

$$h_i = \sum_{j=1}^{n} a_j A_{ji}$$

$j$: row # in the matrix
$i$: column # in the matrix

This can be written concisely as,

$$h = Aa$$
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 $a_0, a_1, a_2,...$ and $h_0, h_1, h_2,...$ converge (to limits $x^*$ and $y^*$)

- $a_1=(((1 \times 0)+(1 \times 0)+(1 \times 1)+(1 \times 0)),
(1 \times 1)+(1 \times 0)+(1 \times 0)+(1 \times 0)),
((1 \times 1)+(1 \times 1)+(1 \times 0)+(1 \times 0)),
((1 \times 1)+(1 \times 1)+(1 \times 1)+(1 \times 1))) = (1,1,2,4)$

Normalize it: $(1/(1+1+2+4), 1/(1+1+2+4), 2/(1+1+2+4), 4/(1+1+2+4)) = (1/8, 1/8, 1/4, 1/2)$

- $a_1=(1/8, 1/8, 1/4, 1/2)$ (authority values after the first iteration)

After the normalization:

- $h_1=(7/22, 6/22, 5/22, 4/22)$ (hub values after the first iteration)
Implementing Topic Search using HITS

- Step 1.
  - Constructing a focused subgraph based on a query

- Step 2.
  - Iteratively calculate the authority value and hub value of the page in the subgraph

Step 1. Constructing a focused subgraph (root set)

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

Root set
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>Nodes</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>Nodes</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>2/8</td>
</tr>
<tr>
<td>P4</td>
<td>4/22</td>
<td>4/8</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)

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 considered as a valid answer, however I do not recommend this
  - This will take much longer time to compute the values

- Random walk style implementation will be more straightforward for this problem
  - Why? Your dataset is a sparse graph
  - 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

   Section 1. MapReduce

   Section 2. Apache Spark
3. Distributed Computing Models for Scalable Batch Computing

Section 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?

- **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
MapReduce

- **MapReduce** provides an abstraction that allows engineers to perform simple computations while hiding the details of **parallelization**, **data distribution**, **load balancing** and **fault tolerance**

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

---

MapReduce Example 1
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
99999 # 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)
99999
V020
320 # wind direction (degrees)
1 # quality code
N
```

The first entries for 1990

```
% ls raw/ 1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```
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` "\n"
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

```bash
% ./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
  1. Types of input/output key-values
  2. The map function
  3. The reduce function

- Optional components
  1. Combiner
  2. Partitioner
  3. InputFormat/OutputFormat

Visualizing the way the MapReduce works (1/3)

Sample lines of input data

```
0067011990999991950051507004... 9999999N9 + 00001 +99999999999...
0043011990999991950051512004... 9999999N9 + 00221 +99999999999...
0043011990999991950051518004... 9999999N9-00111 +99999999999...
0043012650999991949032412004... 0500001N9 + 01111 +99999999999...
0043012650999991949032418004... 0500001N9 + 00781 +99999999999...
```

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

```
(0,  0067011990999991950051507004...9999999N9 + 00001+ 99999999999...)
(106, 0043011990999991950051512004...9999999N9 + 00221+ 99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111 1+ 99999999999...)
```

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

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:

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

This is the final output
MapReduce Example 2

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?

```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

- Run the MapReduce application

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
/usr/joe/wordcount/input /usr/joe/wordcount/output

$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop, 1
Hello 2
World! 1
World, 1
hadoop. 1
to 1
```

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));
    }
}
```
MapReduce Data Flow

MapReduce data flow with a single reducer

Split 0

Split 1

Split 2

Map

Map

Map

sort

sort

sort

copy

Merge

Reduce

Part 0

HDFS Replication

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

Design your map and reduce function to perform following data processing. Instagram has 1 billion users currently.

Find the 10 Instagram users who has the highest number of followers for each age group. The range of age groups is a year. (e.g. 25, 26, 27, 28 …)

The data is formatted as follows:

{InstagramUserID, TAB, date_of_birth, TAB, number_of_followers, 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 InstagramUserIDs are unique.

(1) Mapper

**Input:** <dummy key (e.g. file offset number), a line of the input file>

**Functionality:**
- Tokenize the string and calculate the age
- Generate an output

**Output:** <age, [InstagramUserID, number_of_followers]>

(2) Reducer

**Input:** <age, a list of [InstagramUserID, number_of_followers]>

**Functionality:**
- Scan the list of values and identify top 10 Instagram users with highest number of followers

**Output:** <age, a list of users>

---

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>

**Functionality:**
- Create multiple data structures (HashMap: local_top10_25, local_top10_26... ) to store the local top 10 information (user id and number of followers) per age
- Tokenize the string and calculate the age
- If this user is considered as one of the local top 10 until this point in the age group, update local_top10_x.
- After the input split is completely scanned, generate output with local_top10_x.

**Output**: <age-x, local_top10_x>

(2) Reducer

**Input:** <age-x, a list of [local_top10_x]>

**Functionality:**
- Scan the list of values and identify top 10 users with highest number of followers

**Output:** <age-x, a list of users>

- **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

```
public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
    // Create TreeMap(s) for each age. You can maintain a HashMap: age as the key and TreeMap
    // as the value. This example is only for the 1 age. To serve multiple ages, your code should
    // retrieve the corresponding TreeMap for the age.
    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 age and other attributes
        // Your code to evaluate current number of followers
        LocalTop10.put(number_of_followers, new Text(your_value));
        if (repToRecordMap.size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
        }
    }

    protected void cleanup(Context context) throws IOException,
    InterruptedException {
        // Output our ten records to the reducers with an age as the key
        for (Text t : repToRecordMap.values()) {
            context.write(age, t);
        }
    }
}
```
Better Answer: Top-N design pattern: More Info

- A map function can generate 0 or more outputs.
- `setup()` and `cleanup()` are called for each Mapper and Reducer “only once”. So, if there are 20 mappers running (10,000 inputs each), the setup/cleanup will be called only 20 times.

Example:
```java
public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    try { while (context.nextKey()) {
        reduce(context.getCurrentKey(), context.getValues(), context);
    } }
    finally { cleanup(context); }
}
```

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?**
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, 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,
    - \[ \text{max}(0, 20, 10, 25, 15) = \text{max}( \text{max}(0, 20, 10), \text{max}(25, 15)) = \text{max}(20, 25) = 25 \]
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?