**FAQs**

- Accommodation request, honor student
- Contact me by Sept. 6, 2019
- Office hour: 9-10AM Fridays
- Many questions about the term project!
  - How to do a successful term project
    1. Term project proposal
    2. Final report/software
    3. Presentation
    4. Software demo
  - Evaluation Criteria (RCC metric):
    - Relevance: Is this a big data problem? Does this team use a big data technology to solve the problem?
    - Challenge: Is this project challenging enough for a senior level CS term project?

---

**Term project proposal**

- Submission: October 31 2019 By 5:00 PM via Canvas
- Title
- Problem formulation
- Your strategy to solve the problem
- Your dataset
- Project timeline (weekly plan and roles)
- Bibliography
- Can we change our topic after the proposal?
  - Yes. But you should let me know before you submit your final report. Otherwise I will grade your final report based on your original proposal.
- Can we discuss with prof. about topic/strategy/systems?
  - **WHY NOT?**

---

**Datasets**

- **Examples:**
  - AWS open data: [https://registry.opendata.aws](https://registry.opendata.aws)
  - Graph/network: [http://snap.stanford.edu](http://snap.stanford.edu)
  - Google public data: [https://cloud.google.com/public-datasets/](https://cloud.google.com/public-datasets/)
  - Kaggle competition: "Note that many of datasets in this site are not large enough." [https://www.kaggle.com/datasets?utm_medium=paid&utm_source=google.com+search&gclid=EAIaIQobChMIg8Ttpsq35AIVARx9Ch1sDgC7EAAYASAAEgJbEfD_BwE](https://www.kaggle.com/datasets?utm_medium=paid&utm_source=google.com+search&gclid=EAIaIQobChMIg8Ttpsq35AIVARx9Ch1sDgC7EAAYASAAEgJbEfD_BwE)
  - How can I explore these datasets?
    - Search if there are "good" datasets available.
    - Retrieve the dataset and see if you can use it.
    - Was it widely used?
    - Was it well-formatted?
    - Was it self-contained?

---

**Topics**

- Introduction to MapReduce
- MapReduce Design Pattern I. Summarization Patterns

---

**Part 1. Large Scale Data Analytics**

**Introduction to MapReduce**
MapReduce Example 2

NCDC data example

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

```
2003
102300 # WSRP weather station identifier
99999 # WSRP weather station identifier
1950011 # Observation date
1200 # Observation time (seconds)
+ 51317 # Latitude (degrees x 1000)
+ 028783 # Longitude (degrees x 1000)
FM-12
+ 0171 # Elevation (meters)
99999
V020
320 # Wind direction (degrees)
1 # Quality code
```

Analyzing the data with Unix Tools (1/2)

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

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

• The complete run for the century took \textit{42 minutes}
• To speed up the processing
  • We need to run parts of the program in parallel
  • Process different years in different processes

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

\begin{verbatim}
007613199999991950051507004...9999999N9 + 00001 +99999999999...
0043011990999991950051512004...9999999N9 + 00221 +99999999999...
0043011990999991950051518004...9999999N9 + 00111 +99999999999...
004312409991950051510004...0000001N9 + 0711 +99999999999...
\end{verbatim}

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

\begin{verbatim}
(1949, \{111, 78\})
(1950, \{0, 22, -11\})
(1950, 0)
(1950, 22)
(1949, 111)
(1950, 22)
\end{verbatim}

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.

\begin{verbatim}
(1949, 0)
(1949, 32)
(1949, -31)
(1949, 111)
(1948, 32)
\end{verbatim}

The output key-value pairs will be sorted by key and grouped by key.

Our reduce function will see the following input:

\begin{verbatim}
(1949, \{111, 78\})
(1949, \{0, 22, -11\})
\end{verbatim}

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

\begin{verbatim}
(1950, 22)
(1949, 78)
\end{verbatim}

This is the final output.

Visualizing the way the MapReduce works (3/3)
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

MapReduce Example 3

Dataset

- Network communication log file

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>220.4.5.20</td>
<td>160.33.1.3</td>
</tr>
<tr>
<td>220.4.5.20</td>
<td>160.33.1.3</td>
</tr>
<tr>
<td>160.33.1.3</td>
<td>79.45.66.9</td>
</tr>
<tr>
<td>160.33.1.3</td>
<td>100.33.1.5</td>
</tr>
<tr>
<td>100.33.1.5</td>
<td>79.45.66.9</td>
</tr>
</tbody>
</table>

- There are two attributes
  - From and To
  - (IP1, IP2): There was a communication from IP1 to IP2

Retrieve records with “220.4.5.20” as a source IP

- Check each record and provide as output only those records containing “220,4,5,20” as a source IP

Retrieve records with “220.4.5.20” as the source

Part 1. Large Scale Data Analytics

Design Pattern 1: Summarization Patterns
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

**Why Design Patterns?**

- Tools for solving problems
- Reusable and providing a general framework
- Developers can spend less time figuring out how she/he is going to solve the problem

**Summarization Patterns**

- Design patterns that produce a top-level, summarized view of your data
  - Helps you to glean insights not available from looking at a localized set of records alone

- Numerical summarizations
  - Max/Min, Count, Average, Median, Standard Deviation

- Inverted index

**Introduction**

- Pattern Description
  - General pattern for calculating aggregate statistical values over data
  - E.g. What is the total annual expense of each customer?

- Intent
  - Group records together by a key field and calculate a numerical aggregate per group to get a top-level view of the larger data set

- Applicability
  - Numerical summarizations should be used when both of the followings are true
    - Numerical data or counting
    - The data can be grouped by specific fields
Reducer Code

```java
public static class MinMaxCountReducer extends Reducer<Text, MinMaxCountTuple, Writable, WritableOutputFormat> {
    // Our output value Writables
    private static Writable outTuple = new Writable();
    public void reduce(Text key, Iterable<MinMaxCountTuple> values, Context context) throws IOException, InterruptedException {
        // Initialize our result
        result = new MinMaxCountTuple();
        // For each record, Map function produces the key-value pair (key, value)
        for (MinMaxCountTuple val : values) {
            // If the value's min is less than the result's min
            if (val.getMin() < result.getMin()) {
                // Set the result's min to value's min
                result.setMin(val.getMin());
            }
            // If the value's max is more than the result's max
            if (val.getMax() > result.getMax()) {
                // Set the result's max to value's max
                result.setMax(val.getMax());
            }
            // Add to our count the number of input values
            sum += val.getCount();
        }
        // Set our user ID as the output key
        outUserId = key.toString();
        // Assume we have developed a class, parse.
        Map<String, String> parsed = < String, String > parsed = parse<R>(userId);
        // Our output key and value
        outTuple.setOutput(outUserId, result);
        // Set the comment count to 1
        outTuple.setCount(1);
        // Emit the shuffled pairs (key, a list of values)
        context.write(outUserId, outTuple);
        // Set the result's max to value's max
        if (val.getMax() > result.getMin()) {
            // Set the result's min to value's min
            result.setMin(val.getMin());
        }
    }
    // Map function For each record, Map function produces the key-value pair (key, value)
    public static class MinMaxCountMapper extends Mapper<Object, Text, Text, MinMaxCountTuple> {
        // Our output key and value Writables
        private static Writable outUserId = new Text();
        private static Writable result = new MinMaxCountTuple();
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // Design your Map function - input, output, functionality
            // For each record, Map function produces the key-value pair (key, value)
            for (Text val : value) {
                // Design your Reduce function - input, output, functionality
                // Calculate the Min, Max and Count of < String, String > parsed = < String, String > parsed = parse<R>(userId);
                // For each record, Map function produces the key-value pair (key, value)
                // Set our count to the number of input values
                result.setCount(sum);
                // Calculate the value's min and max value of < String, String > parsed = < String, String > parsed = parse<R>(userId);
            }
        }
    }
}
```
Numerical Summarization

Example 2. Average

• Calculating an average of Values per user

Map function
For each record, Map function produces the key-value pair (key, value)

Reduce function
For the shuffled pairs (key, [a list of values])
Returns (key, average-value) as the result

Design Pattern 1: Summarization Patterns

Combiner Functions

Hadoop Combiner

• Minimizes the 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

• Hadoop does NOT guarantee how many times it will call combiner for a particular map output record

Example: Find the maximum temperature

• First map produces
  (1950, 0)
  (1950, 20)
  (1950, 10)

• Second map produces
  (1950, 25)
  (1950, 15)

• Input to the reduce function
  (1950, [0, 20, 10, 25, 15])

• Output
  (1950, 25)

If a combiner finds the maximum temperature for each map output:

• First map produces
  (1950, 0)
  (1950, 25)
  (1950, 15)
  → (1950, 25)

• Second map produces
  (1950, 25)
  → (1950, 25)

• Input to the reduce function
  (1950, [0, 20, 10, 25, 15])
  → (1950, [20, 25])

• Output
• Can we use a combiner function for finding average value?

Numerical Summarization
Example 2. Average
• Calculating the Average of Values per user

Map function
emit <userID, (Value, count)>

Combine function
Calculate local average
Returns (count, [a list of (local-average-value, local-count)]) as the result

Reduce function
For the shuffled pairs (userID, [a list of (local-average-value, local-count)])
Returns (userID, global-average-value) as the result

Inverted Index
• Generate Index from a data dataset to map from contents, such as words or numbers
  - Reduces the amount of time to find related items
  - Keyword based search, Web search, and document search
  - e.g. Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment

Structure
Inverted index of StackOverflow links to Wikipedia

Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment.

Map function

For each record, Map function produces the key-value pair (key, value).

Reduce function

For the shuffled pairs (key, [a list of values]) returns (key, value) as the result.

Mapper Code

```java
public static class WikipediaExtractor extends Mapper < Object, Text, Text, Text > {

    private Text link = new Text();
    private Text outkey = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        Map <String, String> parsed = MRDPUtils.transformXmlToMap(value.toString());

        // Grab the necessary XML attributes
        String txt = parsed.get("Body");
        String posttype = parsed.get("PostTypeId");
        String row_id = parsed.get("Id");

        // if the body is null, or the post is a question (1), skip
        if (txt == null || (posttype != null && posttype.equals("1"))) { return; }

        // Unescape the HTML because the SO data is escaped.
        txt = StringEscapeUtils.unescapeHtml(txt.toLowerCase());
        link.set(getWikipediaURL(txt));
        outkey.set(row_id);
        context.write(link, outkey);
    }
}
```

Questions?