FAQS

• PA0 has been posted
  • Feb. 6, 5:00PM via Canvas
  • Individual submission (No team submission)

• Accommodation request, honor student
  • Contact me by Jan 26 2018

• Team for term project
  • 3-4 members

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

```
0057 021310 # USAF weather station identifier
99999 # WBAN weather station identifier
19050101 # observation date
0300 # observation time
+51317 # latitude (degrees x 1000)
+028783 # longitude (degrees x 1000)
FM-12 # elevation (meters)
99999
V020320 # wind direction (degrees)
1 # quality code
```

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
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
```
#!/usr/bin/env bash
for year in all/*
do
echo -ne \"basename \$ year .gz\"

# Extract fields
temp = substr ( $0, 88, 5) + 0;
q = substr ($0, 93, 1);
if (temp != 9999 && q ~ /[01459]/ && temp > max)
max = temp
done

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

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
- Checks 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)
MapReduce

• 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

<table>
<thead>
<tr>
<th>Year</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
<td>111</td>
</tr>
<tr>
<td>1950</td>
<td>22</td>
</tr>
<tr>
<td>1950</td>
<td>-11</td>
</tr>
<tr>
<td>1949</td>
<td>111</td>
</tr>
<tr>
<td>1949</td>
<td>111</td>
</tr>
</tbody>
</table>

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

Visualizing the way the MapReduce works (2/3)

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

<table>
<thead>
<tr>
<th>Year</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
<td>111</td>
</tr>
<tr>
<td>1950</td>
<td>22</td>
</tr>
<tr>
<td>1950</td>
<td>-11</td>
</tr>
<tr>
<td>1949</td>
<td>111</td>
</tr>
<tr>
<td>1949</td>
<td>111</td>
</tr>
</tbody>
</table>

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

Our reduce function will see the following input:

<table>
<thead>
<tr>
<th>Year</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
<td>[111, 111]</td>
</tr>
<tr>
<td>1950</td>
<td>[22, -11]</td>
</tr>
</tbody>
</table>

Visualizing the way the MapReduce works (3/3)

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

<table>
<thead>
<tr>
<th>Year</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
<td>111</td>
</tr>
<tr>
<td>1950</td>
<td>22</td>
</tr>
</tbody>
</table>

This is the final output.

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

- 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

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

---

**Retrieve records with “220.4.5.20” as a source**

Map function

- For each record, check the condition (source == “220.4.5.20”) and produce the key-value pair (“220.4.5.20”, record)

Reduce function

- For the shuffled pairs (“220.4.5.20”, [recordA, recordB, recordC]) returns (“220.4.5.20”, [recordA, recordB]) as the result of eliminating duplications.

---

**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 Numerical summarizations
  - Design patterns that produce a top-level, summarized view of your data

**Introduction**

- Pattern Description
  - General pattern for calculating aggregate statistical values over data
- 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

**Numerical Summarization**

**Example 1. Minimum, Maximum and Count**

- Calculating the Min, Max and Count of **Values** per user

**Map Function**

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Value 1</td>
</tr>
<tr>
<td>A</td>
<td>Value 2</td>
</tr>
<tr>
<td>B</td>
<td>Value 3</td>
</tr>
<tr>
<td>C</td>
<td>Value 4</td>
</tr>
<tr>
<td>D</td>
<td>Value 5</td>
</tr>
<tr>
<td>B</td>
<td>Value 6</td>
</tr>
<tr>
<td>D</td>
<td>Value 7</td>
</tr>
<tr>
<td>D</td>
<td>Value 8</td>
</tr>
<tr>
<td>B</td>
<td>Value 9</td>
</tr>
<tr>
<td>D</td>
<td>Value 10</td>
</tr>
</tbody>
</table>

**Reduce Function**

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

**Map Function**

For each record, Map function produces a key-value pair

- Input: <userid, a list of values (a line of string)> (output: <USRID>, a data structure contains (Key, Max, Count))
- Functionality: Tokenizing the string, fill out the output data structures map on key-value pair

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Value 1</td>
</tr>
<tr>
<td>A</td>
<td>Value 2</td>
</tr>
<tr>
<td>B</td>
<td>Value 3</td>
</tr>
<tr>
<td>C</td>
<td>Value 4</td>
</tr>
<tr>
<td>D</td>
<td>Value 5</td>
</tr>
<tr>
<td>B</td>
<td>Value 6</td>
</tr>
<tr>
<td>D</td>
<td>Value 7</td>
</tr>
<tr>
<td>D</td>
<td>Value 8</td>
</tr>
<tr>
<td>B</td>
<td>Value 9</td>
</tr>
<tr>
<td>D</td>
<td>Value 10</td>
</tr>
</tbody>
</table>

**Reduce Function**

Input: <USRID, a list of data structures>
- Functionality: Fluids over the list and calculate Min, Max and Count
**Mapper Code**

```java
public static class MinMaxCountMapper extends Mapper < Object, Text, Text, MinMaxCountTuple > {
    // Any output key and value writable
    private Text outUserId = new Text();
    private Text outTuple = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String keyString = line.substring(line.indexOf("-01") + 5, line.indexOf("01-"));
        String valueString = line.substring(line.indexOf("-01") + 5, line.indexOf("01-"));

        if (keyString.equals(outUserId.get())) {
            // Data is grouped by a key
            outUser.setCount((int) context.getCounter()
                .get("UserId", keyString));
            outUser.setCount((int) context.getCounter()
                .get("UserId", keyString));

            // Set the minimum and maximum date values to the value
            String userId = line.substring(line.indexOf("-01") + 5, line.indexOf("01-"));
            String value = line.substring(line.indexOf("-01") + 5, line.indexOf("01-"));

            // Get the min and max value of userId
            MinMaxCountTuple result = new MinMaxCountTuple();
            result.setMin((int) context.getCounter()
                .get("UserId", keyString));
            result.setMin((int) context.getCounter()
                .get("UserId", keyString));

            // Set the result's max to value's max
            if (value.getMin() < result.getMin()) {
                result.setMin(value.getMin());
            }

            // Add to our sum for value
            int sum = 0;

            // Set our user ID as the output key
            outTuple.set(userId);

            // Set the minimum and maximum date values to the value
            context.write(outUserId, outTuple);
        }
    }
}
```

**Reducer Code --continued**

```java
public static class MinMaxCountReducer extends Reducer < Text, MinMaxCountTuple, Text, MinMaxCountTuple > {
    // Our output value writable
    private MinMaxCountTuple result = new MinMaxCountTuple();

    public void reduce(Text key, Iterable < MinMaxCountTuple > values, Context context) throws IOException, InterruptedException {
        MinMaxCountTuple result = new MinMaxCountTuple();

        // Initialize our result
        result.setMin(null);
        result.setMax(null);
        int sum = 0;

        for (MinMaxCountTuple val : values) {
            // If the value’s max is more than the result’s max
            if (val.getMax() > result.getMax()) {
                result.setMax(val.getMax());
            }

            // Add to our sum for value
            sum += val.getMin();

            // Set our user ID as the output key
            context.write(outUserId, outTuple);
        }
    }
}
```

**Numerical Summarization Example 2, Average**

- Calculating an average of `Values` per user

**Map Phase**

1. Design your Map function
   - input, output, functionality
   - For each record, Map function produces the key-value pair (key, value)

**Reduce Phase**

2. Design your Reduce function
   - input, output, functionality
   - For the shuffled pairs (key, [a list of values])
   - Returns (key, 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, 10)
  - Output (1950, 10)
- Second map produces (1950, 25) (1950, 15)
  - Input to the reduce function (1950, [0, 20, 10, 25, 15])
  - Output (1950, 25)

- Can we use a combiner function for finding average value?

Numerical Summarization

Example 2. Average

- Calculating the Average of Values per user

<table>
<thead>
<tr>
<th>USER</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>1</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>2</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>3</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>4</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>5</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>6</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>7</td>
<td>Value</td>
</tr>
</tbody>
</table>

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

1. Design your Map function
   - input, output, functionality

2. Design your Combiner function
   - input, output, functionality

3. Design your Reduce function
   - input, output, functionality

Numerical Summarization

Example 2. Average

- Calculating the Average of Values per user

<table>
<thead>
<tr>
<th>USER</th>
<th>USER</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
<tr>
<td>USER</td>
<td>USER</td>
<td>Value</td>
</tr>
</tbody>
</table>

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

1. Design your Map function
   - input, output, functionality

2. Design your Combiner function
   - input, output, functionality

3. Design your Reduce function
   - input, output, functionality

1. Design your Combiner function
   - input, output, functionality

2. Design your Reduce function
   - input, output, functionality
Design Pattern 1: Summarization Patterns

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

Example of an inverted index for document 1 and 2 (D1, and D2)

D1 = "Colorado State University"
{Colorado, State, University}

D2 = "University of Colorado"
{University, of, Colorado}

- Added index
  - Colorado (D1, D2)
  - State (D1)
  - University (D1, D2)
  - of (D2)

Inverted Index of StackOverflow links to Wikipedia

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

1. Design your Map function
   - Input, output, functionality

2. Design your Reduce function
   - Input, output, functionality

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

Inverted Index of StackOverflow links to Wikipedia

Map Function
- If there is any link included in "comment" env 1-click, commentID

Reduce function
- All the commentIDs that referred same wikipedia page will be grouped together
```
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);
    }
}
```