PART 1. LARGE SCALE DATA ANALYTICS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

• PA1 has been posted
  • Sign up for your demo

• Team for term project
  • Please check the “people” tag in your canvas page
Topics

• MapReduce Design Pattern II. Filtering Patterns

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

- The function should be **cumulative** and **associative**

---

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, 20)
  - (1950, 10)
  \[\rightarrow (1950, 20)\]

- Second map produces
  - (1950, 25)
  - (1950, 15)
  \[\rightarrow (1950, 25)\]

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

- Output
  - (1950, 25) \[\rightarrow (1950, 25)\]

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

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

2. Design your Combiner function

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

Combiner function
Calculate local average
Returns (userID, (local-average-value, local-count)) as the result

Reduce function
For the shuffled pairs (userID, [a list of (local-average-values, local-count)])
Returns (userID, global-average-value) as the result
Design Pattern 1: Summarization Patterns
Inverted Index

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}
  Inverted Index
    Colorado – {D1, D2}
    State - {D1}
    University – {D1, D2}
    of – {D2}
Inverted index of StackOverflow links to Wikipedia

Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment
Inverted index of StackOverflow links to Wikipedia

<table>
<thead>
<tr>
<th>CommentID</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Comment-1</td>
</tr>
<tr>
<td>B</td>
<td>Comment-2</td>
</tr>
<tr>
<td>C</td>
<td>Comment-3</td>
</tr>
<tr>
<td>D</td>
<td>Comment-4</td>
</tr>
<tr>
<td>E</td>
<td>Comment-5</td>
</tr>
<tr>
<td>F</td>
<td>Comment-6</td>
</tr>
<tr>
<td>G</td>
<td>Comment-7</td>
</tr>
<tr>
<td>H</td>
<td>Comment-8</td>
</tr>
<tr>
<td>I</td>
<td>Comment-9</td>
</tr>
<tr>
<td>J</td>
<td>Comment-10</td>
</tr>
<tr>
<td>...</td>
<td>...</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 Reduce function**
   - input, output, functionality

**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 = MRDFUtils.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);
    }
}
```

Part 1. Large Scale Data Analytics

Design Pattern 2: Filtering Patterns
This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012

Filtering pattern

- Providing an abstract of existing data

- Many data filtering do NOT require the “reduce” part of MapReduce
  - It does not produce an aggregation

- Known uses
  - Tracking a thread of events
  - Distributed grep
  - Data cleaning
  - Closer view of data
  - Simple random sampling
  - Removing low scoring data
Filtering patterns covered in this class

1. Simple Random Sampling
2. Bloom filter
3. Top 10
4. Distinct

---

**MapReduce Design Patterns II: Filtering Patterns**

1. Simple Random Sampling
Filtering Pattern 1.
Simple Random Sampling

• Each record has an equal probability of being selected
• Useful for sizing down a data set
  • For representative analysis

The structure of the simple filter pattern
Writing a Simple Random Sampling filter

```java
public static class SRSMapper
    extends Mapper < Object, Text, NullWritable, Text > {
    private Random rands = new Random();
    private Double percentage;
    protected void setup(Context context) throws IOException, InterruptedException {
        // Retrieve the percentage that is passed in via the configuration
        // like this: conf.set("filter_percentage", .5);
        // for .5%
        String strPercentage = context.getConfiguration().get("filter_percentage");
        percentage = Double.parseDouble(strPercentage) / 100.0;
    }

    public void map( Object key, Text value, Context context) throws IOException, InterruptedException {
        if(rands.nextDouble() < percentage) {
            context.write(NullWritable.get(), value); // otherwise, drop it.
        }
    }
}
```

Stratified Sampling with MapReduce

- Implementing the Stratified Sampling
- Proportion allocation
  - The size of portion in each stratum is taken in proportion to the size of the stratum
- e.g. Sampling from Olympic athletes
  - With two strata, soccer vs. marathon participants
    1. Create the strata
    2. A stratum is made of elements belonging to the same class
    3. Apply Simple Random sampling to each stratum
MapReduce Design Patterns II: Filtering Patterns

2. Bloom Filter

Filtering Pattern 2. Bloom Filter

- Checking the membership of a set

- Known uses
  - Removing most of the non-membership values
  - Prefiltering a data set for an expensive set membership check
What is a Bloom Filter?

• Burton Howard Bloom in 1970

• **Probabilistic data structure** used to test **whether a member is an element** of a set

• **Strong space advantage**

Building a Bloom filter

• $m$
  • The number of bits in the filter

• $n$
  • The number of members in the set

• $p$
  • The desired false positive rate

• $k$
  • The number of different hash functions used to map some element to one of the $m$ bits with a uniform random distribution
Building a Bloom filter

- \( m = 8 \)
  - The number of bits in the filter
- \( n = 3 \)
  - The number of members in the set \( T = \{5, 10, 15\} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)

<table>
<thead>
<tr>
<th>Initial bloom filter</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{5, 10, 15\} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(5) = 7 \), \( h_2(5) = 5 \), \( h_3(5) = 5 \)

<table>
<thead>
<tr>
<th>Initial bloom filter</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>After ( h_1(5) = 7 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_2(5) = 5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_3(5) = 5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(10) = 6 \), \( h_2(10) = 7 \), \( h_3(10) = 2 \)

<table>
<thead>
<tr>
<th>After encoding 5</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>After ( h_1(10) = 6 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_2(10) = 7 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_3(10) = 2 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(15) = 6 \), \( h_2(15) = 7 \), \( h_3(15) = 7 \)

<table>
<thead>
<tr>
<th>After encoding 5 and 10</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>After ( h_1(15) = 5 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_2(15) = 1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>After ( h_3(15) = 7 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Applying a Bloom filter

• Is 5 part of set \( T \)?
  • \( h1(5), h2(5), h3(5) \)'th bits are 1
    • 5 is probably a part of set \( T \)

After encoding 5, 10 and 15
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h1(5) = 7 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h2(5) = 5 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h3(5) = 5 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Applying a Bloom filter

• Is 8 part of set \( T \)?
  • \( h1(8), h2(8), h3(8) \)
  • 8 is NOT a part of set \( T \)

After encoding 5, 10 and 15
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h1(8) = 0 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h2(8) = 3 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Check \( h3(8) = 0 \)
\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Applying a Bloom filter

• Is 9 part of set $T$?
• $h1(9), h2(9), h3(9)$
• 9 is **NOT** a part of set $T$

| After encoding 5, 10 and 15 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h1(9) = 3$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h2(9) = 5$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h3(9) = 1$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

After encoding 5, 10 and 15

| After encoding 5, 10 and 15 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h1(7) = 7$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h2(7) = 1$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| Check $h3(7) = 7$ | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

Applying a Bloom filter

• Is 7 part of set $T$?
• $h1(7), h2(7), h3(7)$'th bits are 1
  • 7 is **probably** a part of set $T$
Hash functions

- $k$ Hash functions
  - Uniform random distribution in $[1..m)$

- Cryptographic hash functions
  - MD5, SHA-1, SHA-256, Tiger, Whirlpool...

- Murmur Hashes (non-cryptographic), Jenkins, Java hashCode(), Spooky, CityHash

Avalanche Effect

- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the $(x, y)$ cell’s color represents the probability that flipping $x$’th bit of input will result of $y$th bit being flipped in the output
Avalanche Effect

- Each bit should have **50% chances** to change if you **change 1 bit** of the input

- Avalanche Diagram
  - A grid such that the \((x,y)\) cell’s color represents the probability that flipping \(x\)’th bit of input will result of \(y\)th bit being flipped in the output

\[
\text{False positive rate (1/2)}
\]

\[
fpr = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k
\]

- \(m=\text{number of bits in the filter}\)
- \(n=\text{number of elements}\)
- \(k=\text{number of hashing functions}\)

The false positive probability \(p\) as a function of number of elements \(n\) in the filter and the filter size \(m\). 

https://en.wikipedia.org/wiki/Bloom_filter
False positive rate (2/2)

- A bloom filter with an optimal value for $k$ and 1% error rate only needs 9.6 bits per key.
- Add 4.8 bits/key and the error rate decreases by 10 times
- 10,000 words with 1% error rate and 7 hash functions
  - ~12KB of memory
- 10,000 words with 0.1% error rate and 11 hash functions
  - ~18KB of memory

How big should I make my Bloom Filter?

- Try various values of $k$ and $m$
  - To achieve target false–positive rate \( (1 - e^{-kw/m})^k \)

- Then, how many hash functions should I use?
  - The more hash functions you have
    - the slower your bloom filter
    - The quicker it fills up
  - If you have few hash functions
    - Too many false positives
  - Given an $m$ and an $n$, the optimal value of $k$
    - $(m/n)\ln(2)$
Use cases

- Representing a very large dataset
- Reduce queries to external database
- Google BigTable

Downsides

- False positive rate

- **Hard to remove elements** from a Bloom filter set
  - Setting bits to zero
    - Often more than one element hashed to a particular bits
  - Use a Counting Bloom filter
    - Instead of bit, it stores count of occurrences
    - Requires more memory
Bloom Filtering mapper (checking) [1/2]

```java
public static class BloomFilteringMapper extends Mapper < Object, Text, Text, NullWritable > {
    private BloomFilter filter = new BloomFilter();
    protected void setup(Context context) throws IOException, InterruptedException {
        // Get file from the DistributedCache
        URI[] files=DistributedCache.getCacheFiles(
            context.getConfiguration());
        System.out.println(
            "Reading Bloom filter from: "+files[0].getPath());
        // Open local file for read.
        DataInputStream strm =
            new DataInputStream(new FileInputStream(files[0].getPath()));
        // Read into our Bloom filter.
        filter.readFields(strm);
        strm.close();
    }
```

Bloom Filtering mapper (Checking) [2/2]

```java
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map < String, String > parsed=transformXmlToMap(value.toString());
    // Get the value for the comment
    String comment = parsed.get("Text");
    StringTokenizer tokenizer = new StringTokenizer(comment);
    // For each word in the comment
    while (tokenizer.hasMoreTokens()) {
        // If the word is in the filter,
        // output the record and break
        String word = tokenizer.nextToken();
        if (filter.membershipTest(new Key(word.getBytes()))) {
            context.write(value, NullWritable.get());
            break;
        }
    }
}
```
MapReduce Design Patterns II: Filtering Patterns

3. Top 10 (Top K)

Filtering Pattern 3. **Top 10**

- Retrieves a **relatively small number (top K)** of records, according to a ranking scheme in your dataset, no matter how large the data

- Known uses
  - Outlier analysis
  - Selecting interesting data
  - Data summarization
  - Catchy dashboards
The structure of Top 10 pattern

![Diagram of Top 10 pattern]

Mapper [1/2]

```java
public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
    // Stores a map of user reputation to the record
    private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Map<String, String> parsed = transformXmlToMap(value.toString());
        String userId = parsed.get("Id");
        String reputation = parsed.get("Reputation");
        // Add this record to our map with the reputation as the key
        repToRecordMap.put(Integer.parseInt(reputation), new Text(value));
        // If we have more than ten records, remove the one with the lowest rep
        // As this tree map is sorted in descending order, the user with
        // the lowest reputation is the last key.
        if (repToRecordMap.size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
        }
    }

    public static class TopTenReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
        public void reduce(Text key, Iterable<NullWritable> values, Context context) {
            // Do something with key
        }
    }
}
```
**Mapper**

```java
protected void cleanup(Context context)
throws IOException, InterruptedException {
    // Output our ten records to the reducers with a null key
    for (Text t : repToRecordMap.values()) {
        context.write(NullWritable.get(), t);
    }
}
```

**Note:** setup() and cleanup() are called for each Mapper and Reducer. So, if there are 20 mappers running (10,000 inputs each), the setup/cleanup will be called 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);
    }
}
```

**Reducer**

```java
public static class TopTenReducer extends Reducer<NullWritable, Text, NullWritable, Text> {
    private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();
    public void reduce(NullWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text value : values) {
            Map<String, String> parsed = transformXmlToMap(value.toString());
            repToRecordMap.put(Integer.parseInt(parsed.get("Reputation")), new Text(value));
            // If we have more than ten records, remove the one with the lowest rep
            // As this tree map is sorted in descending order, the user with
            // the lowest reputation is the last key.
            if (repToRecordMap.size() > 10) {
                repToRecordMap.remove(repToRecordMap.firstKey());
            }
        }
        for (Text t : repToRecordMap.descendingMap().values()) {
            // Output our ten records to the file system with a null key
            context.write(NullWritable.get(), t);
        }
    }
}
```
Reducer code

```java
public static class DistinctUserReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<NullWritable> values, Context context) throws IOException, InterruptedException {
        // Write the user's id with a null value
        context.write(key, NullWritable.get());
    }
}
```

Questions?