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, 15)
- 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)
  \* (1950, 20)
• Second map produces
  - (1950, 25)
  - (1950, 15)
  \* (1950, 25)
• Input to the reduce function
  - (1950, [0, 20, 10, 25, 15])
  \* (1950, [20, 25])
• Output
  - (1950, 25)
  \* (1950, 25)

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

### Numerical Summarization

**Example 2. Average**

**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

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

**Input, output, functionality**

1. Design your Map function
   - input, output, functionality
2. Design your Combiner function
   - input, output, functionality
3. Design your Reduce function
   - input, output, functionality

### 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)*

- University of Colorado
- State of Colorado
- Colorado State University
- University of Colorado at Boulder
- University of Colorado Denver
- Inverted Index:
  - University - (D1, D2)
  - State - (D2)
  - Colorado State University - (D1, D2)
  - University of Colorado - (D1, D2)
  - Colorado - (D1, D2)
Inverted index of StackOverflow links to Wikipedia

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);
    }
}
```

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

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
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 = 0.0;

    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

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

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

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

- \( m = 8 \)
- \( n = 3 \)
- \( k = 3 \)
- \( h_1(x) = 3x \mod 8 \)
- \( h_2(x) = (2x + 3) \mod 8 \)
- \( h_3(x) = x \mod 8 \)

\[ T = \{ 5, 10, 15 \} \]

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init bloom filter</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_1(5) = 7 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_2(5) = 5 )</td>
<td>1</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(5) = 5 )</td>
<td>1</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>

After encoding 5, 10 and 15

| Bit | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

Applying a Bloom filter

- Is 5 part of set \( T \)?
  - \( h_1(5), h_2(5), h_3(5) \)th bits are 1
  - 5 is probably a part of set \( T \)

After encoding 5, 10 and 15

<table>
<thead>
<tr>
<th>Bit</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check ( h_1(5) = 7 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Check ( h_2(5) = 5 )</td>
<td>1</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>Check ( h_3(5) = 5 )</td>
<td>1</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>

Applying a Bloom filter

- Is 8 part of set \( T \)?
  - \( h_1(8), h_2(8), h_3(8) \)
  - 8 is NOT a part of set \( T \)

After encoding 8, 8 and 8

<table>
<thead>
<tr>
<th>Bit</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check ( h_1(8) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Check ( h_2(8) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Check ( h_3(8) = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Applying a Bloom filter

- Is 9 part of set $T$?
- $h_1(9), h_2(9), h_3(9)$
- 9 is **NOT** a part of set $T$

After encoding 5, 10 and 15

- Is 7 part of set $T$?
- $h_1(7), h_2(7), h_3(7)$
- 7 is **probably** a part of set $T$

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

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

False positive rate (1/2)

$$f_{pr} = (1 - (1 - \frac{1}{m})^k)^n = (1 - e^{-\frac{n}{m}})^k$$

$m$ = number of bits in the filter
$n$ = number of elements
$k$ = 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
  - \( \sim 12 \text{KB of memory} \)
- 10,000 words with 0.1% error rate and 11 hash functions
  - \( \sim 18 \text{KB of memory} \)

How big should I make my Bloom Filter?

- Try various values of \( k \) and \( m \)
  - To achieve target false-positive rate \( (1 - e^{-kn/m})^k \)
- 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 \( k \)
    - \( \left( \frac{m}{n} \right) \ln(2) \)

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

Building Bloom Filter with MapReduce

Running Bloom Filter with MapReduce
Bloom Filtering mapper (checking)

```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
        DataInputStream strm = DistributedCache.getCacheFiles( context.getConfiguration() ).get(0);
        // Open local file for read.
        DataInputStream stdin = new DataInputStream( new FileInputStream( getPath( context ) ) );
        // Read into our Bloom filter.
        filter.readFields( stdin );
        stdin.close();
    }

    private void map( Object key, Text value, Context context ) throws IOException, InterruptedException {
        // If the word is in the filter,
        // output the record and break
        String word = tokenizer.toString();
        if ( filter.membershipTest( word.getBytes() ) ) {
            context.write( new Text( word ), new Text( value.toString() ) );
        } else {
            // Filter the word
            while ( tokenizer.hasMoreTokens() ) {
                String comment = tokenizer.nextToken();
                if ( filter.membershipTest( comment.getBytes() ) ) {
                    context.write( new Text( comment ), new Text( value.toString() ) );
                }
            }
        }
    }

    private void reduce( Key key, Iterable < Text > values, Context context ) throws IOException, InterruptedException {
        // Get the value for the comment
        String comment = parseXmlToMap( () );
        // If the word is in the filter,
        // output the record and break
        String word = tokenizer.toString();
        if ( filter.membershipTest( word.getBytes() ) ) {
            context.write( new Text( word ), new Text( value.toString() ) );
        } else {
            // Filter the word
            while ( tokenizer.hasMoreTokens() ) {
                String comment = tokenizer.nextToken();
                if ( filter.membershipTest( comment.getBytes() ) ) {
                    context.write( new Text( comment ), new Text( value.toString() ) );
                }
            }
        }
    }
}
```

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
Reducer

```java
public static class DistinctUserReducer extends Reducer<NullWritable, Text, NullWritable, Text> {
    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());
    }
}
```