No Cell-phones in the class.

If you need to use a laptop, please sit in the back row. I will ask you to turn off your laptop if it seems to be distracting to others.

FAQs

• Why do we need to “shuffle” for the MapReduce jobs?
  • P.A.O
    • Aug. 31, 5:00PM via Canvas
    • If you are not assigned the "port range", contact GTA immediately
  • Team for term project
    • 3-4 members
    • Aug. 30 (TOMORROW), 5:00PM via Canvas
    • Looking for team members? Post your advertisement on Piazza!

Topics covered in this lecture

• MapReduce Design Pattern II. 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 "c-reduce" part of MapReduce
  • It does not produce an aggregation

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

1. Simple Random Sampling
2. Bloom filter
3. 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

The structure of the simple filter pattern

To manage the mapreduce jobs without Reducer (Map-only jobs):
1. You can declare job.setNumReduceTasks(0);
2. You can use IdentityReducer

If you set the number of reduce tasks as 0, there will be no sorting and shuffling. IdentityReducer will still do sorting and shuffling.

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, women and men participants
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, women and men 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. Top 10 (Top K)

Filtering Pattern 2. 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

Mapper [1/2]

```java
public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
    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());
        }
    }
}
```
**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);
    }
}
```

**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);
        }
    }
}
```

NullWritable is a special type of Writable. It has a zero-length serialization.

**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());
    }
}
```

**Filtering Pattern 3: 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**
- **n = 3** target set T = {5, 10, 15}
- **k = 3**
  - **h1(x) = 3x mod 8**
  - **h2(x) = (2x + 3) mod 8**
  - **h3(x) = x mod 8**

**Building a Bloom filter**

- **m = 8**
- **n = 3** target set T = {5, 10, 15}
- **k = 3**
  - **h1(x) = 3x mod 8**
  - **h2(x) = (2x + 3) mod 8**
  - **h3(x) = x mod 8**

**Applying a Bloom filter**

- Is 5 part of set T? If yes, show how to check if 5 is probably a part of set T.
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 \( T \)

| \( h_1(8) = 7 \) | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| \( h_2(8) = 4 \) | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| \( h_3(8) = 0 \) | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

Check \( h_1(8) = 0 \)
Check \( h_2(8) = 3 \)
Check \( h_3(8) = 1 \)

After encoding 5, 10 and 15

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 \( T \)

| \( h_1(9) = 3 \) | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| \( h_2(9) = 5 \) | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| \( h_3(9) = 1 \) | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

After encoding 5, 10 and 15

Applying a Bloom filter

- Is 7 part of set \( T \)?
  - \( h_1(7), h_2(7), h_3(7) \)th bits are 1
  - 7 is probably a part of set \( T \)

| \( h_1(7) = 7 \) | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| \( h_2(7) = 1 \) | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| \( h_3(7) = 7 \) | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

After encoding 5, 10 and 15

Hash functions

- A Hash functions
  - Uniform random distribution in \([1..m)\]
- Cryptographic hash functions
  - MD5, SHA-1, SHA-256, Tiger, Whirlpool
- Must satisfy cryptographic hash function properties
  - Preimage resistance
    - For a random value \( h \) chosen by an honest party, it's very costly for an attacker to find any value \( m \) such that \( \text{hash}(m) = h \)
  - Second-preimage resistance
    - For a random value \( m_1 \) chosen by an honest party, it's very costly for an attacker to find any value \( m_2 \neq m_1 \) such that \( \text{hash}(m_1) = \text{hash}(m_2) \)
  - Collision resistance
    - It's very costly for an attacker to find any pair of values \( m_1 \neq m_2 \) such that \( \text{hash}(m_1) = \text{hash}(m_2) \)
- 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 (Black: 0%, red: 100%)

Perfect bit independence

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
False positive rate (1/2)
\[ fp_r = \left(1 - \frac{1}{m}\right)^{kn} = (1 - e^{-kn/m})^k \]

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

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 \( (1 - e^{-kn/m}) \)

- 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

Building Bloom Filter with MapReduce

- Loading data
- Bloom filter mapper
- Big vector generation
- Reduction phase
Running Bloom Filter with MapReduce

Bloom Filter Mapper

Input Split
Load filter
Bloom filter test
Output File
Discarded
Maybe
No

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 {
        System.out.println("Reading Bloom filter from: "+DistributedCache.getCacheFiles(context.getConfiguration())[0].getPath());
        DataInputStream strm = new DataInputStream(new FileInputStream(DistributedCache.getCacheFiles(context.getConfiguration())[0].getPath()));
        filter.readFields(strm);
        strm.close();
    }
```

Hadoop Distributed cache is a mechanism supported by Hadoop mapreduce framework where the users can broadcast small or moderate size files to all of the worker nodes.

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 {
        System.out.println("Reading Bloom filter from: "+DistributedCache.getCacheFiles(context.getConfiguration())[0].getPath());
        DataInputStream strm = new DataInputStream(new FileInputStream(DistributedCache.getCacheFiles(context.getConfiguration())[0].getPath()));
        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());
    String comment = parsed.get("Text");
    StringTokenizer tokenizer = new StringTokenizer(comment);
    while (tokenizer.hasMoreTokens()) {
        String word = tokenizer.nextToken();
        if (filter.membershipTest(new Key(word.getBytes()))) {
            context.write(value, NullWritable.get());
            break;
        }
    }
}
```

Questions?