Today's topics

- FAQs
- TrustRank and Spam mass
- MapReduce design patterns
- Filtering patterns

FAQs

- PA2 dataset is available
- GTA will announce the instruction.
- Revised Deadline: March 25th (Friday) 5:00PM via Canvas
  - Firm deadline

- Help session for PA2 has been scheduled
  - March 4 11:00AM ~ noon CSB130
  - The session will be recorded

- Review session for Midterm
  - Thursday (March 3) in the class

Combatting Link Spam

- Detecting and eliminating link spam have been critical for search engines
  - Just as it was critical to eliminate term spam in the previous decade

- Detecting particular structures
  - Spam farm
    - One page links to a very large number of pages
    - Each of which links back to it

- Modifying PageRank to lower the rank of link-spam pages automatically
  - TrustRank
  - Spam mass
TrustRank

- TrustRank is a topic-sensitive PageRank
  - “topic” is a set of pages believed to be trustworthy (not spam)
- Develop a suitable teleport set of trustworthy pages
  - Let humans examine a set of pages and decide which of them are trustworthy
- Pick a domain whose membership is controlled
  - University pages
    - .mil, or .gov

Calculating TrustRank (1/2)

- Then the topic-sensitive PageRank for $S$ is the limit of the iteration,
  \[ v' = \beta M v + (1 - \beta) e_S / |S| \]
- $M$ is the transition matrix of the Web, and $|S|$ is the size of set $S$

Calculating TrustRank (2/2)

- Suppose we use $\beta=0.8$, and our trust rank is represented by the teleport set $S=\{B, D\}$
- $\beta M = \begin{bmatrix} 0 & 2/5 & 5/4 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \end{bmatrix}$
- $v' = \begin{bmatrix} 0 \\ 1/10 \\ 0 \\ 1/10 \end{bmatrix} = \beta M v + (1 - \beta) e_S / |S|$
- B and D get a higher PageRank than before

Spam Mass

- Measures the fraction of its PageRank that comes from spam for each page
- For an arbitrary page $p$,
  - PageRank $r$
    - Computing ordinary PageRank
  - TrustRank $t$
    - Computing the TrustRank based on some teleport set of trustworthy pages
- The spam mass
  - $(r - t)/r$
- A negative or small positive spam mass
  - $P$ is probably not a spam page
- Page with high spam mass score
  - Should be eliminated

Example

- Suppose that both the PageRank and TrustRank were computed
- Teleport set was page B and D
  - Which nodes are not the link spams?
  - Is there any link spam?

<table>
<thead>
<tr>
<th>Web Page</th>
<th>PageRank</th>
<th>TrustRank</th>
<th>SpamMass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3/9</td>
<td>54/210</td>
<td>0.229</td>
</tr>
<tr>
<td>B</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
<tr>
<td>C</td>
<td>2/9</td>
<td>38/210</td>
<td>0.168</td>
</tr>
<tr>
<td>D</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
</tbody>
</table>
Example

- Suppose that both the PageRank and TrustRank were computed
- Teleport set was page B and D
- Which nodes are not the link spams?
  - B and D
  - C has lower chance to be the link spam compared to A
- Is there any link spam?

<table>
<thead>
<tr>
<th>Web Page</th>
<th>PageRank</th>
<th>TrustRank</th>
<th>SpamMass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3/9</td>
<td>54/210</td>
<td>0.229</td>
</tr>
<tr>
<td>B</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
<tr>
<td>C</td>
<td>2/9</td>
<td>38/210</td>
<td>0.186</td>
</tr>
<tr>
<td>D</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
</tbody>
</table>

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

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);
        }
    }
}
```

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

MapReduce Design Patterns
Filtering Patterns
2. Bloom Filter

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

- **m = 8, n = 3 target set T = {5, 10, 15}**
- **k = 3**
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(5) = 7, \ h_2(5) = 5, \ h_3(5) = 5 \)
  - Initial bloom filter
    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_1(5) = 7 \)
    | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_2(5) = 5 \)
    | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_3(5) = 5 \)
    | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

- **m = 8, n = 3 target set T = {5, 10, 15}**
- **k = 3**
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(10) = 6, \ h_2(10) = 7, \ h_3(10) = 2 \)
  - Initial bloom filter
    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_3(5) = 5 \)
    | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_1(10) = 6\)
    | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_2(10) = 7 \)
    | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
  - After \( h_3(10) = 2 \)
    | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

Applying a Bloom filter

- Is 5 part of set T?
- \( h_1(5), h_2(5), h_3(5) \)'s bits are 1
- 5 is probably a part of set T
  - Check \( h_1(5) = 7 \)
    | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
  - Check \( h_2(5) = 5 \)
    | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
  - Check \( h_3(5) = 5 \)
    | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
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 \( T \)

Check \( h_1(8) = 0 \) \[ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \]
Check \( h_2(8) = 3 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]
Check \( h_3(8) = 9 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]

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 \( T \)

Check \( h_1(9) = 3 \) \[ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \]
Check \( h_2(9) = 5 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]
Check \( h_3(9) = 1 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]

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 \( T \)

Check \( h_1(7) = 7 \) \[ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \]
Check \( h_2(7) = 1 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]
Check \( h_3(7) = 7 \) \[ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \]

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)

False positive rate (1/2)

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

- \( m \) = 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
Use cases

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

Building and running Bloom Filtering

Input Split → Load filter → Bloom Filter Building → Output Split

Bloom Filter Mapper

Input Split → Load filter → Bloom Filter Building → Output Split

Bloom Filter Mapper

Writing a Bloom filter [2/2]

```java
for (FileStatus status : fs.listStatus(inputFile)) {
    BufferedReader rdr = new BufferedReader(new InputStreamReader(fs.open(status.getPath())));
    System.out.println("Trained Bloom filter with " + numMembers + " entries.");
    System.out.println("Serializing Bloom filter to HDFS at " + numMembers);
    filter.add(new Key(status.getPath()));
    ++numElements;
}
```

Writing a Bloom filter mapper [1/2]

```java
public static class BloomFilterDriver {
    public static void main(String[] args) throws Exception {
        // Parse command line arguments
        Path inputfile = new Path(args[0]);
        int numMembers = Integer.parseInt(args[1]);
        float falsePosRate = Float.parseFloat(args[2]);
        Path bffile = new Path(args[3]);
        // Calculate our vector size and optimal k value based on approximations
        int vectorSize = getOptimalBloomFilterSize(numMembers, falsePosRate);
        int nbHash = getOptimalNBHash(numMembers, vectorSize);
        // Create new Bloom filter
        BloomFilterFilter = new BloomFilter(vectorSize, nbHash, Hash.MURMUR_HASH);
        // Open file for read
        String line = null;
        int numMembers = 0;
        FileSystem fs = FileSystem.get(new Configuration());
    }
}
```

Writing a Bloom filter mapper [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());
        // Open local file for read.
        DataInputStream strm = new DataInputStream(new FileInputStream(files[0].getPath()));
        // Read into our Bloom filter.
        filter.readFields(strm);
        strm.close();
    }
}
```

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
Writing a Bloom Filtering mapper  

```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()) {
        String word = tokenizer.nextToken();
        if (filter.membershipTest(new Key(word.getBytes()))) {
            context.write(value, NullWritable.get());
            break;
        }
    }
}
```

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
  - Catchy dashboards

The structure of Top 10 pattern

```
Input Split  Filter Mapper  Local Top 10
Input Split  Filter Mapper  Local Top 10
Input Split  Filter Mapper  Local Top 10
Input Split  Filter Mapper  Local Top 10
```

Mapper

```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
        if (repToRecordMap.size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
        }
    }
    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>
{
    // Stores a map of user reputation to the record
    private TreeMap<Integer, Text> repToRecordMap = new TreeMap<>();

    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
            if (repToRecordMap.size() > 10) {
                repToRecordMap.remove(repToRecordMap.firstKey());
            }
        }

        for (Text t : repToRecordMap.descendingMap().values()) {
            context.write(NullWritable.get(), t);
        }
    }
}
```

Filtering Pattern 4. Distinct

- You have data that contains similar records and you want to find a unique set of values

```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());
    }
}
```

Combiner

- How can you improve the performance of previous MapReduce software using a Combiner?

Mapper Code

```java
public static class DistinctUserMapper extends Mapper<Object, Text, Text, NullWritable> {
    private Text outputUserId = new Text();
    public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
        Map<String, String> parsed = transformXmlToMap(value.toString());
        // Get the value for the UserId attribute
        String userId = parsed.get("UserId");
        // Set our output key to the user’s id
        outputUserId.set(userId);
        // Write the user’s id with a null value
        context.write(outputUserId, NullWritable.get());
    }
}
```
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

Patterns discussed in this section
1. Generating data
2. External source input
3. Partition pruning

Modify the way data is loaded on disk
- Approach 1: Configuring how contiguous chunks of output are generated from blocks in HDFS
  - OutputFormat
- Approach 2: Configuring how records appear in the map phase
  - RecordReader

Customizing input and output
- Do we always want to load or store data the way Hadoop MR does out of the box?
  - Injecting data from original source without storing data in HDFS
  - Feeding the MapReduce output to the next process

FAQs
- Term project phase II: your topic!
  - Data?
  - Services?
  - Analysis?
    - Use the analysis we have discussed in the class
      - Similarity measures, Bloomfilter, simple query
    - Use the analysis you have learned in the stat class
      - Linear regression
      - Trend analysis
Roles of InputFormat in Hadoop

1. Validate the input configuration for the job (i.e., checking that the data is there).

2. Split the input blocks and files into logical chunks of type InputSplit, each of which is assigned to a map task for processing.

3. Create the RecordReader implementation to be used to create key/value pairs from the raw InputSplit. These pairs are sent one by one to their mapper.

Methods of the InputFormat abstract

- `getSplits()`: retrieves the configured input using the JobContext object
- `getLocations()`: returns the list of hostnames where the input split is located
- `createRecordReader()`: Called by framework and generates RecordReader

RecordReader (1/2)

- Generates WritableComparable key and Writable value
- An object-oriented way to present information to a mapper
- `TextInputFormat` grabs each line
- Example
- `<xhtml version="1.0" encoding="UTF-8">
  <quiz>
    <question>
      Who was the forty-second president of the U.S.A.?
    </question>
    <question>
      Which of the following states is a border state?
    </question>
    <!-- More questions here... -->
  </quiz>
</xhtml>`

RecordReader (2/2)

- Reads Bytes from the input source
- Generates WritableComparable key and Writable value
- Example
- `TextInputFormat` grabs each line
- `<"xml version="1.0" encoding="UTF-8">
  <quiz>
    <question>
      Who was the forty-second president of the U.S.A.?
    </question>
    <question>
      Which of the following states is a border state?
    </question>
    <!-- More questions here... -->
  </quiz>
</xhtml>`

Accessing your input file in MapReduce

- STEP 1. Validates the input for the job by checking whether all of the input paths exist
- STEP 2. Splits each input file logically based on the total size of the file in bytes
  - Block size is the upper bound
    - E.g. 160MB in HDFS will generate three blocks
      - 2 x 64MB and 1x38MB
  - STEP 3. Each map task will be assigned exactly one of these input splits
  - STEP 4. RecordReader will generate key/value pairs for Mapper input

Methods of the RecordReader (abstract)

- `initialize()`: initializes
- `getCurrentKey()` and `getCurrentValue()`: returns the current key and value
- `nextKeyValue()`: calls next key/value
- `getProgress()`: returns the progress
- `close()`: closes the reader
Schema on read

- `InputSplit` represents a `byte-oriented` view of the split
- `RecordReader` prepares data for a mapper
  - Only the `RecordReader` maintains the schema

Storing data in an External DB

- MapReduce job is not restricted to storing data to HDFS
- MapReduce can do a parallel bulk write
  - Your storage should be able to handle the large number of connections from the many tasks

- E.g. `DBOutputFormat<K, DBWritable, V>`
  - Objects that read from/written to a database should implement `DBWritable`
  - If we have the following table in the database:

```sql
CREATE TABLE MyTable
(
  counter INTEGER NOT NULL,
  timestamp BIGINT NOT NULL,
);
```

Writing your output to a DB (2/2)

```java
public void write(PreparedStatement statement) throws SQLException {
  statement.setInt(1, counter);
  statement.setLong(2, timestamp);
}
```

```java
public void readFields(ResultSet resultSet) throws SQLException {
  counter = resultSet.getInt(1);
  timestamp = resultSet.getLong(2);
}
```

```
**** PreparedStatement is an object that represents a precompiled SQL statement.

```java
private void write(PreparedStatement statement) throws SQLException {
  statement.setInt(1, 153833.00);
  statement.setObject(2, 110592);
}
```

Writing your output to a DB (1/2)

```java
public class MyWritable implements Writable, DBWritable { // Some data
  private int counter;
  private long timestamp;

  // WritableWrite() implementation
  public void write(DataOutput out) throws IOException {
    out.writeInt(counter);
    out.writeLong(timestamp);
  }

  // WritableReadFields() implementation
  public void readFields(DataInput in) throws IOException {
    counter = in.readInt();
    timestamp = in.readLong();
  }
}
```

I/O Pattern 1: Generating Data

- Generates a lot of data from scratch
  - This pattern does not load data
- Use cases:
  - Generating random data
  - Generating artificial data as part of a benchmark
    - TeraGen/TeraSort and DFSIO
- This pattern is map-only
Structure

- The InputFormat creates the fake splits from nothing
- The RecordReader takes its fake split and generates random records
- The IdentifyMapper is used to just write the data out as it comes in

Identity Mapper

- Implements Mapper<K,V, K,V>
- Identity Mapper takes input key/value pair and returns without any processing
- Other implementations of Mapper
  - InverseMapper, TokenCountMapper, ChainMapper... Etc.

Identity Reducer

- Implements Reducer<K,V, K,V>
- Performs no reduction, writing all input values directly to the output.

- What is the difference between Identity Reducer and 0 reducer?
  - Identity reducer still sort and shuffle output data from the mappers
    - No aggregation

I/O Pattern 1: Generating Data: Example

- Goal
  - Generates random StackOverflow data
  - Take a list of 1,000 words and make random blurbs

Driver code

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    int numMapTasks = Integer.parseInt(args[0]);
    int numRecordsPerTask = Integer.parseInt(args[1]);
    Path wordList = new Path(args[2]);
    Path outputDir = new Path(args[3]);
    Job job = new Job(conf, "RandomDataGenerationDriver");
    job.setJarByClass(RandomDataGenerationDriver.class);
    job.setNumReduceTasks(0);
    job.setInputFormatClass(RandomStackOverflowInputFormat.class);
    RandomStackOverflowInputFormat.setNumMapTasks(job, numMapTasks);
    RandomStackOverflowInputFormat.setNumRecordPerTask(job, numRecordsPerTask);
    RandomStackOverflowInputFormat.setRandomWordList(job, wordList);
    TextOutputFormat.setOutputPath(job, outputDir);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(NullWritable.class);
    System.exit(job.waitForCompletion(true) ? 0 : 2);
}
```
InputSplit Code

```java
public static class FakeInputSplit extends InputSplit implements Writable {
    public void write(DataOutput arg0) throws IOException {
    }
    public void readFields(DataInput arg0) throws IOException {
    }
    public long getLength() throws IOException, InterruptedException {
        return 0;
    }
    public String[] getLocations() throws IOException, InterruptedException {
        return new String[0];
    }
}
```

continued

```java
public RecordReader<Text, NullWritable> createRecordReader(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {
    RandomStackOverflowRecordReader rr = new RandomStackOverflowRecordReader();
    rr.initialize(split, context);
    return rr;
}
```

InputFormat code

```java
public static class RandomStackOverflowInputFormat extends InputFormat<Text, NullWritable> {
    public static final String NUM_MAP_TASKS = "random.generator.map.tasks";
    public static final String NUM_RECORDS_PER_TASK = "random.generator.num.records.per.map.task";
    public static final String RANDOM_WORD_LIST = "random.generator.random.word.file";
    public static void setNumMapTasks(Job job, int i) {
        job.getConfiguration().setInt(NUM_MAP_TASKS, i);
    }
    public static void setNumRecordPerTask(Job job, int i) {
        job.getConfiguration().setInt(NUM_RECORDS_PER_TASK, i);
    }
    public static void setRandomWordList(Job job, Path file) {
        DistributedCache.addCacheFile(file.toUri(), job.getConfiguration());
    }
}
```

I/O Pattern 2: External Source Output

- Writing MapReduce output to a nonnative location
- In a MapReduce approach, the data is written out in parallel

The Structure of the external source output pattern

![Diagram of I/O Pattern 2: External Source Output]

- The OutputFormat verifies the output specification of the job configuration prior to job submission
- The RecordWriter writes all key/value pairs to the external source
Example

- Writing the results to a number of Redis instances
- Redis is an open-source, in-memory, key-value store
- Redis provides Jedis (Java client of Redis)
- A Redis hash is a map between string fields and string values
- Similar to a Java hashmap

continued

public void checkOutputSpace( JobContext job) throws IOException {
    String hosts = job.getConfiguration().get( REDIS_HOSTS_CONF);
    if (hosts == null || hosts.isEmpty()) {
        throw new IOException( REDIS_HOSTS_CONF + " is not set in configuration.");
    }
    String hashKey = job.getConfiguration().get( REDIS_HASH_KEY_CONF);
    if (hashKey == null || hashKey.isEmpty()) {
        throw new IOException( REDIS_HASH_KEY_CONF + " is not set in configuration.");
    }
    Job job = jobConfJob();
    getOutputCommitter().set( REDIS_HOSTS_CONF, hosts);
    getOutputCommitter().set( REDIS_HASH_KEY_CONF, hashKey);
    public static class RedisHashRecordWriter extends RecordWriter {
      
      public void map( Object key, Text value, Context context) throws IOException, InterruptedException {
          // For each jedis instance, disconnect it for (Jedis jedis) 
          jedis.disconnect();
      }
      
      public void close( TaskAttemptContext context) throws IOException, InterruptedException {
          // For each jedis instance, disconnect it for (Jedis jedis = jedisMap.get( key.hashCode())) {
          jedis.disconnect();
      }

OutputFormat Code

public static class RedisHashOutputFormat extends OutputFormat {
    public static final String REDIS_HASH_KEY_CONF = "mapred.redishashoutputformat.hashkey";
    public static final String REDIS_HOSTS_CONF = "mapred.redishashoutputformat.hosts";
    public static void setRedisHashOutputFormat(Job job, String hosts) {
        job.setConfiguration().set( REDIS_HOSTS_CONF, hosts);
        job.setConfiguration().set( REDIS_HASH_KEY_CONF, hashKey);
    }
    
    public static class RedisHashRecordWriter extends RecordWriter {
        
        public void map( Text key, Text value) throws IOException, InterruptedException {
            // Get the jedis instance that this key/ value pair will be written to Jedis
            Jedis jedis = jedisMap.get( key.hashCode());
            if (jedis != null) {
                // Write the key/ value pair
                jedis.hset( hashKey, key.toString(), value.toString());
            } else {
                throw new IOException( "Can not get jedis instance for key: " + key.toString());
            }
            // Context output key and values
            outkey.set( key.toString());
            outvalue.set( value.toString());
            context.write( outkey, outvalue);
        }

Mapper Code

public static class RedisOutputMapper extends Mapper {
    private Text outkey = new Text();
    private Text outvalue = new Text();
    public void map( Object key, Text value, Context context) throws IOException, InterruptedException {
        // Set our output key and values
        outkey.set( keyHashCode);
        outvalue.set( reputation);
        context.write( outkey, outvalue);
    }
}
Driver Code

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Path inputPath = new Path(args[0]);
    String hosts = args[1];
    Job job = new Job(conf, "Redis Output");
    job.setJarByClass(RedisOutputDriver.class);
    job.setMapperClass(RedisOutputMapper.class);
    job.setReducerClass(RedisLastAccessInputSplit.class);
    TextInputFormat.setInputPaths(job, inputPath);
    job.setInputFormatClass(RedisInputFormat.class);
    job.setOutputFormatClass(RedisOutputFormat.class);
    RedisHashOutputFormat.setRedisHosts(job, hosts);
    job.setOutputValueClass(Text.class);
    job.setNumReduceTasks(0);
    job.setJarByClass(RedisOutputDriver.class);
    job.setOutputFormatClass(RedisHashOutputFormat.class);
    RedisHashOutputFormat.setRedisHashKey(job, hashKey);
    job.setOutputValueClass(Text.class);
    job.waitForCompletion();
    System.exit(code);
}
```

The Structure of the partition pruning pattern

I/O Pattern 3: Partition Pruning

- Configures the way the framework picks input splits and drops files from being loaded into MapReduce based on the name of the file
- Partitions data by a predetermined value
- Use cases
  - Organizing your data based on your analysis patterns
  - Change analytics? Or, change data input format?

Writing InputSplit

```java
public static class RedisLastAccessInputSplit extends InputSplit implements Writable {
    private String location = null;
    private List<String> hashKeys = new ArrayList<String>();
    public RedisLastAccessInputSplit() {
        // Default constructor for reflection
    }
    public RedisLastAccessInputSplit(String redisHost) {
        this.location = redisHost;
    }
    public void addHashKey(String key) {
        hashKeys.add(key);
    }
    public void removeHashKey(String key) {
        hashKeys.remove(key);
    }
    public List<String> getHashKeys() {
        return hashKeys;
    }
    public void readFields(DataInput in) throws IOException {
        this.location = in.readUTF();
        int numKeys = in.readInt();
        hashKeys.clear();
        for (int i = 0; i < numKeys; i++) {
            hashKeys.add(in.readUTF());
        }
    }
    public void write(DataOutput out) throws IOException {
        out.writeUTF(this.location);
        out.writeInt(hashKeys.size());
        for (String key : hashKeys) {
            out.writeUTF(key);
        }
    }
    public long getLength() throws IOException, InterruptedException {
        return 0;
    }
    public String getLocations() throws IOException, InterruptedException {
        return new String[] { location };
    }
}
```

Writing InputFormat

```java
public static class RedisLastAccessInputFormat extends InputFormat {
    private static String REDIS_SELECTED_MONTHS_CONF = "mapred.redislastaccessinputformat.months";
    private static final HashMap<String, Integer> MONTHS_FROM_STRING = new HashMap<String, Integer>();
    private static final HashMap<String, String> MONTH_TO_INST_Map = new HashMap<String, String>();
    public static final String REDIS_SELECTED_MONTHS_CONF = "mapred.redislastaccessinputformat.months";
    static {
        // Initialize month to Redis instance map
        // Initialize month 3 character code to integer
        for (int i = 0; i < 3; i++) {
            MONTHS_FROM_STRING.put("0" + i, i);
            MONTH_TO_INST_Map.put(0 + i, "0" + i);
        }
    }
    public static class RedisLastAccessInputSplit extends InputSplit {
        private String location = null;
        private List<String> hashKeys = new ArrayList<String>();
        public RedisLastAccessInputSplit() {
            // Default constructor for reflection
        }
        public RedisLastAccessInputSplit(String redisHost) {
            this.location = redisHost;
        }
        public void addHashKey(String key) {
            hashKeys.add(key);
        }
        public void removeHashKey(String key) {
            hashKeys.remove(key);
        }
        public List<String> getHashKeys() {
            return hashKeys;
        }
}
```
public List < InputSplit > getSplits ( JobContext job) throws IOException {  
    String months = job.getConfiguration().get(REDIS_SELECTED_MONTHS_CONF);
    if (months == null || months.isEmpty()) { throw new IOException(REDIS_SELECTED_MONTHS_CONF + " is null or empty.");

    // Create input splits from the input months
    HashMap < String, RedisLastAccessInputSplit > instanceToSplitMap = new HashMap < String, RedisLastAccessInputSplit >();
    for (String month : months.split(',')) {
        String host = MONTH_TO_INST_MAP.get(month);
        RedisLastAccessInputSplit split = instanceToSplitMap.get(host);
        if (split == null) {
            split = new RedisLastAccessInputSplit(host);
            split.addHashKey(month);
            instanceToSplitMap.put(host, split);
        } else {
            split.addHashKey(month);
        }
    }

    return new ArrayList < InputSplit > (instanceToSplitMap.values());
}

public static class RedisLastAccessRecordReader extends RecordReader < RedisKey, Text > {
    …
}

continued

if (split == null) {
    split = new RedisLastAccessInputSplit( host);
    split.addHashKey( month);
    instanceToSplitMap.put( host, split);
} else {
    split.addHashKey( month);
}
return new ArrayList < InputSplit > (instanceToSplitMap.values());
public RecordReader < RedisKey, Text > createRecordReader (InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {
    return new RedisLastAccessRecordReader();
}
public static class RedisLastAccessRecordReader extends RecordReader < RedisKey, Text > {
    …
}