PART 1. LARGE SCALE DATA ANALYTICS

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

FAQs

• PA0 submission is open
  • Feb. 6, 5:00PM via Canvas
  • Late submission: Feb. 8, 5:00PM via Canvas

• PA1 has been posted
  • Feb. 21, 5:00PM via Canvas
  • Individual submission (No team submission)

Topics

• PA1
• MapReduce Design Pattern IV. Join Patterns

Programming Assignment 1

• Creating N-gram profile for a Wikipedia Corpus
• Due: 5:00PM Feb 21 2018
• Objectives
  • Basic features of Hadoop distributed file system and MapReduce
  • Creating NGram profiles using Hadoop MapReduce

Programming Assignment 1

• N-Gram
  • A contiguous sequence of N items from a given sequence of text or speech
  • “We analyze large datasets”
  • 1-grams (aka unigram)
  • 2-grams (aka bigram)
    • (__, We), (We, analyze), (analyze, large), (large, datasets), (datasets, __)
    • https://books.google.com/ngrams/info

• Requirements
  1. extracting all distinct unigrams
  2. computing the frequency of each unigram per article and also over the corpus
  3. ranking the unigrams based on these frequencies
Structure of PA1: Profile 1

- Extract unigrams
- Sort and eliminate duplicates
- Output Part A

Handling input data

- Ignore tense, and gender of words
  - "He" and "she" should be different unigrams
  - "was" and "is" should be different unigrams
- Do not distinguish the plural words
  - "name" and "names" should be considered as different unigrams
- Hyphenated words
  - "well-known" should be considered as 1 unigram
- Convert all the upper cases to the lower case first, if needed
  - "Ngram" and "Ngram" should be considered as 1 unigram
- Do not eliminate stop words (the most common words in a language)
  - Include "a" and "the" in your unigram profiles

Input Data

- Format

```
Title_of_Article-1------DocumentId-1------Best_of_Article-1
NEWLINE
Title_of_Article-2------DocumentId-2------Best_of_Article-2
```

Programming Assignment 1

Profile 1

A list of unigrams that occurred at least once in the entire corpus (1G dataset). The unigrams must be sorted in (ascending) alphabetical order. You should eliminate duplicates. The output should be generated using MapReduce. You may store the output in multiple files.

Profile 2

A list of unigrams and their frequencies within the target article. Your software must generate this profile per article. Your list should be grouped by the Document ID (see page 1), and sorted (in descending order) on the frequency of the unigram within the article. This output should be generated using MapReduce. Output may be stored in multiple files.

Profile 3

A list of unigrams and their frequencies within the target corpus. The list of unigrams should be sorted in descending order on the frequency of the unigram within the corpus (1G dataset). This output should be generated using MapReduce. Output may be stored in multiple files.

Output data format

- Profile 1
  - Ngram-A NEWLINE
  - Ngram-B NEWLINE
  - Ngram-C NEWLINE

- Profile 2
  - Document-ID TAB ngram TAB frequency NEWLINE

- Profile 3
  - Document-ID TAB ngram TAB frequency NEWLINE
Part 1. Large Scale Data Analytics

Design Pattern 4: Join Patterns

Join Patterns

- Data is all over the place
- "Joins" allow users to create a smaller reference set or filter out or select a dataset to discover interesting relationships across datasets
- Joining a terabyte of data onto another terabyte dataset could require up to two terabytes of bandwidth!
  - That’s before any actual join logic can be done!

1. Reduce Side Join Pattern
2. Replicated Join Pattern
3. Composite Join Pattern
4. Cartesian Product Pattern

Composite Join

- Joins very large datasets together
- And if the datasets are sorted by foreign key
- No shuffle and sort needed
- Each input dataset must be partitioned and sorted in a specific way and divided into the same number of partitions
Joining process (Inner Join)- Within a group

Joining process (Inner Join)- Between groups

Example: Composite user comment join

- Preconditions
  - User and comment datasets should be preprocessed by MapReduce
  - CompositeInputFormat will be used
  - Output should use the TextOutputFormat
    - Key: userID
    - Value: userXML or comment XML
    - Hadoop KeyValueToFileOutputFormat can parse these information

- Note: Since CompositeInputFormat uses Text objects as the key for comparisons
  - Align your keys as Text objects (rather than LongWritable)
  - e.g. "12345" comes before "2"

Driver Code [1/2]

```java
public static void main(String[] args) throws Exception {
    Path userPath = new Path(args[0]);
    Path commentPath = new Path(args[1]);
    Path outputDir = new Path(args[2]);
    String joinType = args[3];
    JobConf conf = new JobConf("CompositeJoin");
    conf.setMapperClass(CompositeMapper.class);
    conf.setReducerClass(CompositeReducer.class);
    conf.setInputFormatClass(CompositeInputFormat.class);
    // The input format scans through the left dataset and identifies matching in the right dataset.
    // It generates input value, [key from the left dataset, item from the right dataset]
    // Set the input format class to a CompositeInputFormat class.
    // The CompositeInputFormat will parse all of our input files and output
    // records to our mapper.
    conf.setInputFormat(CompositeInputFormat.class);
```
Mapper Code

```java
public static class CompositeMapper extends MapReduceBase implements Mapper<Text, TupleWritable, Text, Text> {
    public void map(Text key, TupleWritable value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
        // Get the first two elements in the tuple and output them
        output.collect((Text) value.get(0), (Text) value.get(1));
    }
}
```

Cartesian Product

- Pairs every record from multiple inputs
- Applicability
  - You want to analyze relationships between all pairs of records
  - e.g. Correlation analysis

Performance Analysis

- Cartesian product requires a massive data space
  - To compute a Cartesian product between N records and M records
  - \( N \times M \) data points are created
  - Requires many Map slots and a very long time

Example: Cartesian Product

- Comments comparison
  - Inspects a pair of comments and determines how similar they are
  - If they are similar enough, the pair is stored in the output file
- During the setup phase
  - `getInputSplits`
    - Creates the cross-product of the input splits and sets into a list of CompositeInputSplits
HDFS Block Size vs. Hadoop Input Split

- **Block Size**
  - Physical location where the data has been stored (default size of the HDFS block is 128MB)
  - All blocks of the file are of the same size except the last block
  - HDFS distributes these blocks (chunks) across the cluster
  - Achieves more parallelism and fault tolerance (with replication)

- **Input Split**
  - Logical representation of block
  - More or less than a Block Size
  - Used during data processing in MapReduce program or other processing techniques
  - InputSplit does not contain actual data but a reference to the data
  - Split acts as a broker between the block and mapper
  - During MapReduce execution, Hadoop scans through the blocks and creates InputSplits and assigns them to individual mappers

Example of Using Input Split

- **1.28GB File**
  - Divided into 10 blocks (128MB)
  - InputFormat.getSplits is responsible for generating the input splits
  - Default: creates one input split for each HDFS block

Case 1. If input split is not specified
- HDFS block size will be the split size
- 10 mappers are initialized to load the file
- Each mapper loads one block

Case 2. If the start and end positions of the records are not in the same block?
- InputSplit provides the start and end positions (offsets)
- Mapper is going to load the block of data according to start and end offset values

From Input Splits to Mapper tasks

- Hadoop’s Interface InputFormat<K, V>
  - RecordReader<K, V>
  - InputSplit()"}

- Validate the input specification of the job
- Split-up the input file(s) into logical InputSplits
  - Assign to an individual Mapper

- Provide the RecordReader implementation to the user to glean input records from the logical InputSplit for processing Mapper

RecordReader<K, V>

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>close() Close this RecordReader Implementation to future operations.</td>
</tr>
<tr>
<td>String</td>
<td>createKey() Create an object of the appropriate type to be used as a key</td>
</tr>
<tr>
<td>String</td>
<td>createValue() Create an object of the appropriate type to be used as a value</td>
</tr>
<tr>
<td>Long</td>
<td>getPos() Returns the current position in the input.</td>
</tr>
<tr>
<td>Boolean</td>
<td>getKey() Returns the current key/value pair from the input for processing.</td>
</tr>
</tbody>
</table>

InputSplit

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>getLength() Get the total number of bytes in the data of the InputSplit.</td>
</tr>
<tr>
<td>String</td>
<td>getLocations() Get the list of hostnames where the input split is located.</td>
</tr>
</tbody>
</table>
Input Format Code

```java
public static class CartesianInputFormat extends FileInputFormat {
    public static final String LEFT_INPUT_FORMAT = "cart.left.inputformat";
    public static final String RIGHT_INPUT_FORMAT = "cart.right.inputformat";
    public static final String LEFT_INPUT_PATH = "cart.left.path";
    public static final String RIGHT_INPUT_PATH = "cart.right.path";

    private static void setLeftInputInfo(JobConf job, Class <? extends FileInputFormat> inputFormat, String inputPath) {
        job.setLeftInputPath(inputPath);
        job.setLeftInputFormat(inputFormat.getCanonicalName());
    }

    private static void setRightInputInfo(JobConf job, Class <? extends FileInputFormat> inputFormat, String inputPath) {
        job.setRightInputPath(inputPath);
        job.setRightInputFormat(inputFormat.getCanonicalName());
    }

    public static void setInputPaths(JobConf job, Class <? extends FileInputFormat> inputFormat) throws IOException {
        // Get the left input splits
        leftSplits = inputFormat.getSplits(job);
        for (InputSplit leftSplit : leftSplits) {
            job.setLeftInputPath(leftSplit.getPath());
            job.setLeftInputFormat(leftSplit.getFormatClass().getCanonicalName());
        }

        // Get the right input splits
        rightSplits = inputFormat.getSplits(job);
        for (InputSplit rightSplit : rightSplits) {
            job.setRightInputPath(rightSplit.getPath());
            job.setRightInputFormat(rightSplit.getFormatClass().getCanonicalName());
        }
    }

    public static void setRightInputInfo(JobConf job, Class <? extends FileInputFormat> inputFormat, String inputPath) {
        // Create a new instance of the input format
        FileInputFormat newInputFormat = (FileInputFormat) ReflectionUtils.newInstance(inputFormatClass, conf);
        conf.setJobName("CartesianMapReduce", newInputFormat);

        // Set the input path for the left data set
        newInputFormat.setInputPaths(leftConf, LEFT_INPUT_PATH, LEFT_INPUT_FORMAT, newInputFormatClass);
        // Set the input path for the right data set
        newInputFormat.setInputPaths(rightConf, RIGHT_INPUT_PATH, RIGHT_INPUT_FORMAT, newInputFormatClass);

        // Create a new CompositeInputFormat instance
        compositeInputFormat = new CompositeInputFormat(newInputFormat);
        job.setInputFormatClass(compositeInputFormat.getClass());
    }

    // Create our CompositeInputSplits, size equal to
    // left.length * right.length
    CompositeInputSplit[] returnSplits = new CompositeInputSplit[leftSplits.length * rightSplits.length];
    returnSplits.length = 0;
    for (InputSplit leftS : leftSplits) {
        for (InputSplit rightS : rightSplits) {
            returnSplits[returnSplits.length++] = new CompositeInputSplit(leftS, rightS);
        }
    }

    // Return the composite splits
    LOG.info("Total splits to process: " + returnSplits.length);
    return returnSplits;
}
```

Record reader

```java
public static class CartesianRecordReader < K1, V1, K2, V2 > implements RecordReader < Text, Text > {
    // Record reader to get key value pairs
    private RecordReader leftReader = null, rightReader = null;
    // Store configuration to re-create the right record reader
    private FileInputFormat leftFIF;
    private JobConf leftConf;
    private InputSplit leftSplit;
    private Reporter leftReporter;
    // Helper variables
    private K1 key;
    private V1 value;
    private K2 key2;
    private V2 value;
    private boolean gotLeft = true, alldone = false;
```
public CartesianRecordReader CompositeInputSplit split, JobConf conf, Reporter reporter) throws IOException {
    this.rightConf = conf;
    this.rightIS = split.get(1);
    this.rightRR = reporter;
    // Create left record reader
    FileInputFormat leftFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName("org.apache.hadoop.mapreduce.lib.input.FileInputFormat").getConstructor(JobConf.class).newInstance(conf));
    leftRR = leftFIF.getRecordReader(split.get(0), conf);
    return this;
}

public boolean next(Text key, Text value) throws IOException {
    do {
        // If we are to go to the next left key/value pair
        if (goToNextLeft) {
            // Read the next key value pair, false means no more pairs
            if (!leftRR.next(lkey, lvalue)) { // If no more, then this task is nearly finished
                alldone = true; break;
            }
            // if we aren’t done, set the value to the key and set
            // our flags
            key.set(lvalue.toString());
            goToNextLeft = alldone = false;
            // Reset the right record reader
            this.rightRR = this.rightFIF.getRecordReader(this.rightIS, this.rightConf, thin.right Reporter);
            continue;
        } else { // If we are done, set the key to the key and set
            // our flags
            key.set(rvalue.toString());
            goToNextLeft = alldone = false;
            // Reset the right record reader
            this.rightRR = this.rightFIF.getRecordReader(this.rightIS, this.rightConf, thin.right Reporter);
            return true;
        }
    } while (!job.isComplete());
    return false;
}

Driver Code

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException, ClassCastException {
    // Configure the join type
    JobConf conf = new JobConf("Cartesian Product");
    conf.setJarByClass(CartesianMapper.class);
    conf.setInputFormat(CartesianInputFormat.class);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    conf.setNumReduceTasks(1);
    // Configure the input format
    CartesianInputFormat.setLeftInputInfo(conf, TextInputFormat.class, args[0]);
    CartesianInputFormat.setRightInputInfo(conf, TextInputFormat.class, args[0]);
    // Configure the join type
    conf.setMapperClass(CartesianProduct.class);
    conf.setOutputValueClass(Text.class);
    conf.setOutputKeyClass(Text.class);
    RunningJob job = JobClient.runJob(conf);
    while (!job.isComplete()) {
        Thread.sleep(1000); // Wait for the job to complete
    }
    System.exit(jobs.isSuccessful() ? 0 : 1);
}
Mapper Code (comparing comments)

```java
public static class CartesianMapper extends MapReduceBase implements Mapper < Text, Text, Text, Text > {
  private Text outkey = new Text();

  public void map(Text key, Text value, OutputCollector < Text, Text > output, Reporter reporter) throws IOException {
    // If the two comments are not equal
    if (!key.toString().equals(value.toString())) {
      String[] leftTokens = key.toString().split("\s");
      String[] rightTokens = value.toString().split("\s");
      HashSet<String> leftSet = new HashSet<>(Arrays.asList(leftTokens));
      HashSet<String> rightSet = new HashSet<>(Arrays.asList(rightTokens));
      int sameWordCount = 0;
      StringBuilder words = new StringBuilder();
      for (String s : leftSet) {
        if (rightSet.contains(s)) {
          words.append(s + ",");
          ++sameWordCount;
        }
      }
      // If there are at least three words, output
      if (sameWordCount > 2) {
        outkey.set(words.toString());
        output.collect(outkey, value);
      }
    } else {
      // If the two comments are equal
      for (String s : leftSet) {
        if (rightSet.contains(s)) {
          words.append(s + ",");
          ++sameWordCount;
        }
      }
      // If there are at least three words, output
      if (sameWordCount > 2) {
        outkey.set(words.toString());
        output.collect(outkey, value);
      }
    }
  }
}
```

Questions?