Hadoop Assignment
Concurrency en Parallel Programmeren 2015,
University of Amsterdam
Version 1.1

Contents

1 Overview

In this assignment, you have to analyze a dataset containing Twitter posts. In particular, you have to perform three tasks:

1. Find the top ten most popular hashtags.
2. Perform sentiment analysis on the English-language tweets, and find the average and standard deviation of the sentiments for each of those top ten hashtags.
3. Investigate retweet and reply behaviour.

There is a small dataset (5k tweets) available on Blackboard. We also have larger datasets available in `/projects/cpp2015` on the cluster. Using them for your experiments is optional, but you might find them useful. Remember, though: the cluster is a shared resource. Don't run large numbers of unnecessary experiments!

For several parts of the assignment, you'll have to make your own decisions about what to do – for example, which characters you will accept as part of hash tags, and how you will detect identical retweets. Justify your decisions in your report!

2 Implementation

You should already have the framework from the tutorial; you should simply build on top of it, and implement these analysis tasks using MapReduce with Hadoop.

Remember: first, develop your code locally, and make sure it works there, and then make sure it works on the cluster, where you should run your experiments for the report.

3 Tag Counter

First, you have to find the top ten tags in the dataset. You should do the following:

1. Work out how to provide entire tweets (including all the associated information) as input to your mapper. Take a look at the format of the example dataset file, and think about what you could set the `textinputformat.record.delimiter` configuration option to, to make MapReduce split up the input appropriately.
2. In your `Mapper` class, extract the text of each tweet, and detect if a line contains any hash tags.
3. If the line contains one or more hash tags, emit them to the reducers using the appropriate keys/-values.
4. In your `Reducer` class, count the number of hash tags.
5. Afterwards, examine the final output, and determine which hash tags are the most popular. **Attention:** You shouldn't implement anything else in MapReduce (e.g. a sorting algorithm) to get the top ten tags. You can simply write a script or some code which reads the output files, after you downloaded the results. (Why don't you have to merge the output files for this?).

When you're done, it might be a good idea to make a copy of your code, before you move onto the next task.

3.1 Sentiment Analysis

For the second part of the assignment, you will need to further modify both the `Mapper` and `Reduce` classes.

In natural language processing, language identification is the problem of determining which natural language is used in a text, while sentiment analysis aims to determine the attitude of a person according the her/his writings.

To perform language detection, you will use the JLangDetect API (`https://github.com/melix/jlangdetect`). Import the UberLanguageDetector package:

```
import me.champeau.ld.UberLanguageDetector;
```

Then, to detect the language of a String variable named `text`, you can use:

```
String lang = UberLanguageDetector.getInstance().detectLang(text);
```

If the language in the variable `text` is in English, the `lang` variable will be set to `'en'`.

To perform sentiment analysis on English text, you will use the Stanford Natural Language Processing API (`http://nlp.stanford.edu/`). To use this API you'll need to import the necessary packages in your class:

```
import java.util.Properties;
import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.neural.rnn.RNNCoreAnnotations;
import edu.stanford.nlp.sentiment.SentimentCoreAnnotations;
import edu.stanford.nlp.trees.Tree;
import edu.stanford.nlp.util.CoreMap;
```

To detect the sentiment of a String variable named `text`, you can use the following method:

```
private int findSentiment(String text) {
  Properties props = new Properties();
  props.setProperty("annotators", "tokenize, ssplit, parse, sentiment");
  props.put("parse.model", parseModelPath);
  props.put("sentiment.model", sentimentModelPath);
  StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

  int mainSentiment = 0;
```

```
    if (text != null && text.length() > 0) {
  int longest = 0;
  Annotation annotation = pipeline.process(text);
  for (CoreMap sentence : annotation
  .get(CoreAnnotations.SentencesAnnotation.class)) {
    // 'AnnotatedTree' is 'SentimentAnnotatedTree' in newer versions
    Tree tree = sentence
    .get(SentimentCoreAnnotations.AnnotatedTree.class);
    int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
    String partText = sentence.toString();
    if (partText.length() > longest) {
      mainSentiment = sentiment;
      longest = partText.length();
    }
  }
    }

    return mainSentiment;
  }
```

This method returns an integer that reflects the sentiment of a text. The higher the number, the more positive the sentiment of the text.

Hint: Loading the models is very slow. You can dramatically improve the performance of this function by creating the pipeline in the class constructor instead.

It might be helpful to know that the standard deviation can be iteratively (value-by-value) calculated like this, after you've already calculated the average (mean):

```
  sd = sd + Math.pow(val - mean, 2);
```

Importantly, in this method, you need to specify the `parseModelPath` and `sentimentModelPath` variables. The first variable gives the path to the `englishPCFG.ser.gz` file, which is used to perform parsing of English-language text. The second gives the path to the `sentiment.ser.gz` file, used for sentiment detection.

For local testing/use, you can simply download these files from Blackboard. However, on the cluster, you'll need to make sure that the files are available to all the mappers. We've already put them on the HDFS, in `/projects/cpp2015`.

To make them available, you just have to use the `DistributedCache`, which is provided by the MapReduce framework to cache files (text, archives, etc) needed by applications. To use it, you have to add the files you need to the cache before running your job:

```
job.addCacheFile(new URI("/projects/cpp2015/sentiment.ser.gz"));
```

Of course, you'll need to import the URI class too. Once you've done that, the files you've cached will be available in the directory which your mapper runs in (so you can just open them by their filename).

This path doesn't exist on your own computer, so to make this work properly on both your own machine and the cluster, you'll need to check whether your code is running locally, or on the cluster using MapReduce 2.0 (*yarn*). You can obtain the framework you're running under by checking the value of the `mapreduce.framework.name` configuration option, which will be `yarn` when running on the cluster, or `local` if running locally:

```
if (conf.get("mapreduce.framework.name").equals("yarn"))
```

To summarize, to complete this assignment, you'll need to:

1. Add the `englishPCFG.ser.gz` and `sentiment.ser.gz` files to the list of cached files.
2. Use the mappers to detect if a tweet contains any hash tags, and if it does, detect the language of the tweet.
3. If the language is English, perform the sentiment analysis, and emit appropriate keys and values to the reducers.
4. Finally, use the reducers to calculate the average (mean) sentiment and the standard deviation, for each hash tag.

4 Retweets and Replies

For the final part of the assignment, you need to investigate the main two types of engagement on Twitter: retweets, and replies. Remember, the dataset is from 2009, so no metadata is available for this, only the message text. Take a look through the example data, and you should find plenty of examples.

Retweets: When people retweet (using RT), it usually means they think the content of the tweet is interesting or useful, and so they think their followers would also benefit from seeing it. By retweeting, they say "take a look," and expose the tweet to all their followers, giving the tweet a greater reach.

Reply: When a tweet is replied to (using @), it suggests that the tweet has resonated enough with someone that it sparks a conversation, or encourages someone to share it with their followers. You'll find that most "replies" aren't actually replies, but just references to another user. We don't expect you to try and distinguish these; you can just assume that they're all replies (if they're not retweets).

Given the popularity of these two activities, it would be interesting to explore how they are used. You should write code, using MapReduce, which answers the following questions (in approximate order of probable difficulty):

- How many tweets are there for each type of reaction (retweets, and replies)?
- When do most retweets happen (what times of day), and when do most replies happen? (**Optional**: Is this different for different languages?)
- What are the top 10 retweeted messages, and who are they from?
- Who are the top 5 users who are replied to, and which users send the most replies to each of them?
- Who are the top 5 users whose messages are retweeted, and who retweeted each of them the most?
- **Optional**: Are tweets with a certain sentiment more likely to generate more retweets?
- **Optional**: Do those top 10 retweeted messages get significantly more replies? (This one might be quite tricky.)

These are not easy, and you will probably need to implement several different mapper/reduce classes. For some of these questions, you might want to run a second MapReduce job, using the output of your first one. For others, you might find it most efficient to use a script or some code on your own computer to calculate the final answer.

You'll probably want to work on a separate copy of your code for this; remember to keep a copy of your code from the other tasks.

Finally: In the provided data sets (especially the smaller ones), you will find that there aren't a unique set of top 5 users, or top 10 messages. You can just pick arbitrarily (or use a slightly different number, if you'd prefer). Just show that you've implemented something which *can* answer the question, and explain why!

5 Report

You are required to hand in both a report and your code:

1. Your source code, for all of the three tasks. You can hand in one version of the code for each task, or (ideally) one single piece of code that implements all three parts of the analysis. Please only hand in the *source code*. We don't want your `target` folder or your output files, but your source code **must** be complete, including any scripts you may have written/used.
2. A short report, of no more than 3 pages.

Your source code should be comprehensively commented, so that we can understand exactly what you're doing. Assume that we already know how MapReduce works, but you should still explain how your code interacts with MapReduce in general.

Your report should contain the following sections:

1. Introduction: Very briefly (no more that 2 paragraphs), describe the major components of MapReduce with Hadoop.
2. Implementation: Describe your approach for implementing each part of the Twitter dataset analysis, from a high-level point of view.
3. Results:
   (a) A table with the top ten hash tags that you found, their occurrences, their average sentiment and their standard deviation.
   (b) A graph showing speedup measurements when using 1, 2, 4 and 8 mappers, for at least the sentiment analysis task.
   (c) Tables or graphs which present your answers to the questions about retweets and replies.
4. Conclusion:
   (a) Discuss your results!
   (b) Make sure to talk about the benefit, if any, of adding more mappers. (If you didn't observe any speed up by adding more mappers, explain why!)
   (c) Explain what parts of this problem you think are well-suited to Hadoop/MapReduce, and which (if any) are not.

5.1 Experiments

Since you are sharing the cluster, it is possible that your measurements may be influenced by other activities. With this in mind, when you measure speedup, you should be sure to perform multiple experiments, and present your findings in a statistically-sound manner. Be sure to graph the speedup using error bars. Remember, you can calculate speedup (the relative performance improvement when executing a task) using:

$$S_p = \frac{T_1}{T_p}$$

Where T_1 is the execution time when using a single mapper, and T_p is the execution time when using p mappers.

6 Grading

Deadline: See Blackboard. Remember, you have to hand in all your scripts with your code, so you *must* have already made sure you can run all the experiments you need.
Grading: Tag Counter: 4/10, Sentiment Analysis: 3/10, Retweets and Replies: 3/10.
To obtain full points, your code must work correctly, be well-commented and tidy, and produce correct results. In your report, you will have to include all the requested graphs and results, and provide everything requested in Section 5.