Overview

As you have seen by now, creating a search engine can require storing and computing on lots of data. While it may be feasible to run a search engine for a subset of Wikipedia on a single commodity machine, this is not a scalable solution. Modern information retrieval systems are distributed systems, with the workload and storage spread out across many nodes.

This project is designed to give you a taste of distributed computing. Our tool for this purpose will be Amazon Elastic MapReduce, and our goal will be to analyze a large (i.e. many gigabytes) Google Books dataset.

1 Amazon Elastic MapReduce and Hadoop

Amazon Elastic MapReduce (EMR) is one of a suite of services offered by Amazon Web Services (AWS). The terminology can be confusing here, so take note of the following definitions:

- **MapReduce** – a programming model (and corresponding implementation from Google) for performing parallelizable processing tasks on large clusters of computers. “Map” refers to a master node distributing the processing among worker nodes, whereas “reduce” refers to combing the results from worker nodes into the solution data.

- **Hadoop** – Apache’s open-source implementation of the MapReduce model for distributed computing.

- **Amazon Elastic Compute Cloud (EC2)** – a suite of services designed for pre-configured servers in Amazon’s cloud.

- **Amazon Simple Storage Service (S3)** – your input data, output data, and the code for your MapReduce jobs will all be stored on this AWS service.

Amazon Elastic MapReduce is an interface for running and managing Hadoop jobs on EC2 instances. There are quite a few steps to getting Elastic MapReduce up and running. First you will need to create an Amazon Web Services account. Navigate to [http://aws.amazon.com](http://aws.amazon.com) and click “Sign Up Now”. Then follow the instructions for creating an account.

Elastic MapReduce is not free (or, TANSTAAFI[1]). However, the course staff has managed to obtain AWS credit granted for educational purposes. We will distribute a separate promotion code to each group.

---

[1] [http://en.wikipedia.org/wiki/There_ain%27t_no_such_thing_as_a_free_lunch](http://en.wikipedia.org/wiki/There%27ain%27t_no_such_thing_as_a_free_lunch)

Now that you have signed up for AWS and you have AWS credit, there are some configuration steps for Elastic MapReduce. We recommend that you thoroughly follow the EMR ‘Getting Started Guide’ at [http://docs.aws.amazon.com/ElasticMapReduce/2009-03-31/GettingStartedGuide/Welcome.html](http://docs.aws.amazon.com/ElasticMapReduce/2009-03-31/GettingStartedGuide/Welcome.html). This guide walks you through the installation process for the EMR command-line interface, which is a useful way to manage your EMR jobs. There are a lot of steps in this tutorial; make sure to follow them carefully. You are welcome to use the EMR Management Console in lieu of the command-line interface, although the TA staff finds the command-line interface to be relatively straightforward.

The tutorial above describes two mechanisms for running EMR jobs—Hive job flows and streaming job flows. Hive is a high-level SQL-like language built on top of Hadoop that can allow you to perform processing tasks with just a few lines of code. Streaming job flows can be written in several supported languages, including CS 158’s favorite programming language, Python. They are based on streaming output on the standard input and standard output. Refer to Amazon’s documentation for details on how to use Python for a streaming job flow. Presuming that you have set up the EMR command-line interface correctly, this boils down to running a command such as the following:

```
./elastic-mapreduce --create --stream
   --name "Word Count Job"
   --mapper s3n://158elasticmapreduce/sample/wordCount.py
   --input s3n://158elasticmapreduce/sample/fullCollection.dat
   --output s3n://mys3bucket/output
   --reducer aggregate
```

This command creates a new streaming job named ‘Word Count Job’. It indicates that the job should execute the Python program `wordCount.py`, which reads from the standard input and writes to the standard output. The input data file is `fullCollection.dat`, and the output will be written to a directory called `output` in the S3 bucket called `mys3bucket`. The word count code in `wordCount.py` is nothing fancy—you can take a look at the complete program as part of the EMR tutorial linked above.

If you would like to use Hive rather than a streaming job flow, the following tutorial should give you a great headstart: [http://aws.amazon.com/articles/5249664154115844](http://aws.amazon.com/articles/5249664154115844). This shows you how to start a Hive job flow, and then ssh into the master node. You can then run Hive queries directly from the master node. This is arguably much easier than writing a streaming job—although perhaps less familiar than Python scripting.

### 2 Data

You will be working with data compiled from Google Books that has already been made available on Amazon S3. The format of the dataset is described here: [http://aws.amazon.com/datasets/8172056142375670](http://aws.amazon.com/datasets/8172056142375670). Each row of the data is an n-gram (i.e. a tuple of n consecutive words from a natural language source) and associated metadata. The data is separated by language—in addition to English, there are n-gram datasets produced from books in Chinese, German, and Hebrew, among other languages. For this assignment,
however, we will stick to the American English datasets. You will have to make use of the unigram, bigram,
and trigram data. The unigram data has about 300 million rows while the trigram data has more than 12
dillion—both much larger than the subset of Wikipedia that you are working with for the main project.

Each row of the data contains 1) the n-gram itself, 2) the calendar year in which this n-gram appeared,
3) the number of times the n-gram appeared in books from the corresponding year (“count”), 4) the number
of pages on which the n-gram appeared in this year (“page count”), and 5) the number of books in which
the n-gram appeared in this year (“book count”). The data are in a plaintext format, and contain the five
columns specified above. Here is an example of a few lines taken from the trigram dataset:

<table>
<thead>
<tr>
<th>German tanks during</th>
<th>1959</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>German tanks during</td>
<td>1960</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>German tanks during</td>
<td>1963</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>German tanks during</td>
<td>1969</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

The American English unigram, bigram, and trigram datasets are available at the following S3 bucket
URIs. Note that these files have been compressed by zip.

s3://datasets.elasticmapreduce/ngrams/books/20090715/eng-us-all/1gram/data
s3://datasets.elasticmapreduce/ngrams/books/20090715/eng-us-all/2gram/data
s3://datasets.elasticmapreduce/ngrams/books/20090715/eng-us-all/3gram/data

For testing your job (without EMR), we have produced a partial version of the trigrams file. This file is
available in /course/cs158/data/hadoop, both as a zip file and the corresponding inflated plaintext. Even
though this is just a fraction of the dataset, the plaintext is about 1.8 gigabytes, so we recommend creating
an even smaller test file. Note also that although the file has the .csv suffix (for “comma-separated value”),
the columns are actually tab-separated. Within the n-gram field, the three terms of the trigram are delimited
by spaces. Although the test data should be immensely useful, when you actually run your Hadoop jobs,
you will want to use the complete data by pointing EMR at the URIs above.

3 Report

Using the Google Books n-gram data, answer the following questions. The code that you will need to write
in order to answer these questions is the substantive part of this project. You should write up your answers
in a PDF file called report.pdf and hand it in along with any source code you write. See the sections on
submission (§5) and evaluation (§6) for more details.

1. What is the most common bigram of all time? What is the most common bigram in 1987? How about
   1953?

2. Identify a few words that were coined after 1970. These should be words that never appear before
   1970 but begin to appear (hopefully with some non-negligible frequency) in later decades. You might
   illustrate the increasing usage of the new term by plotting its frequency in the corpus on the y-axis
   against time on the x-axis.
3. Suppose we’re interested in finding trigrams which tend to appear many times on the same page or many times in the same book. What is the trigram that appears at least 10 times with the lowest ratio of page count to total count? What is the trigram that appears at least 10 times with the lowest ratio of book count to total count?

4 Collaboration Policy

You can work on the Course Project in pairs, only with the same team-mate with whom you worked on Part 1 and Part 2 of the Course Project. Only one submission per team is required. Please email cs158tas at cs dot brown dot edu if you have any questions.

5 Submission

In order to hand in, create a directory which contains only of the following files:

- `readme.txt` - this should be identical to the `readme.txt` file that you or your team handed in for the first two parts of the main project.
- `report.pdf` - a PDF file containing your answers to the questions from section 4.
- Any source code that you wrote for running your Hadoop jobs.

Then change to the directory containing these files and run the following command:

```
$ /course/cs158/bin/cs158-handin hadoop
```

6 Evaluation

We will not be running your Hadoop jobs in order to evaluate this project. You should nevertheless make sure that your source code is well-designed and well-documented—we will be reviewing it. The correctness of your work, however, will be judged based on the responses you provide in your report, so make sure that your answers are accurate and cleanly composed!

Have fun!