CprE 419 Lab 2: Text Analysis using Hadoop MapReduce

Department of Electrical and Computer Engineering
Iowa State University

Purpose

The goal is to introduce you to the MapReduce programming model under Hadoop. MapReduce is an abstraction for processing large data sets and has been applied successfully to a number of tasks in the past, including web search. At the end of this lab, you will be able to:

- Write and Execute a program using Hadoop MapReduce
- Write algorithms for data processing using MapReduce
- Apply these skills in analyzing basic statistics of a large text corpus

Submission

Create a zip (or tar) archive with the following and hand it in through Canvas**.**

- A write-up answering questions for each experiment in the lab, if there is any.
- Output for each experiment (screenshots or txt files)
- Commented Code for your program. Include all source files needed for compilation. Name each source file according to the experiment, as appropriate.
- Executable JAR (your program) that produced your output.

MapReduce

MapReduce is a programming model for parallel programming on a cluster. We will be working with the Hadoop implementation of the MapReduce paradigm.

A Program based on the MapReduce model can have several rounds. Each round must have a *map* and a *reduce* method. The input to the program is processed by the map method and it emits a sequence of key and value pairs <k, v>. The system groups all values corresponding to a given key and sorts them. Thus for each key k, the system generates a list of values corresponding to k, and this is then passed on to the reduce method. The output from the reduce methods can then be used as the input to the next round, or can be output to the distributed file system.

A key point is that two mappers cannot communicate with each other, and neither can reducers communicate with each other. Although this restricts what a program can do, it allows for the system to easily parallelize the actions of the mappers and the reducers. The only communication occurs when the output of a mapper is sent to the reducers.

Resources

Take a look at the Example Hadoop Program found on Canvas **– "WordCount.java"**. A MapReduce program typically has at least 3 classes as shown in the example code: A **Driver Class** (in our case, WordCount)**,** a **Map Class** and a **Reduce Class.**

Highly Recommend: you can compile the program using Eclipse, with Maven to import the Hadoop libraries. Find further instructions for development using Eclipse in Lab1.

Don't forget to choose jdk 1.8

Notes

- The MapReduce programs read input from the HDFS and write their output to HDFS.

- Two Hadoop programs running simultaneously cannot have the same output path, although they can share the same input path. Thus, make your output path unique, as otherwise your job may have a conflict with the output of other jobs, and hence fail. Finally make sure that the output directory is empty by deleting its contents. You must do this check every time you re-run the program or it will fail.

- Note that each MapReduce round can have multiple input paths, but only one output path assigned to it. If given an input path that is a directory, MapReduce reads all files within the input directory and processes them.

- You can explore the Hadoop MapReduce API in the link: http://hadoop.apache.org/docs/r2.6.0/api/

- For viewing your job status on your web browser using Hadoop WebUI, use the following link: http://hpc-class.its.iastate.edu:8088/cluster

- For the history of the jobs: http://hpc-class.its.iastate.edu:19888/jobhistory

Experiments 1, (15 points)

The WordCount program counts the number of occurrences of every distinct word in a document. Download the file WordCount.java and compile the program. Make sure the jar is located in your home directory on the cluster. Then run the program as follows:
To run a job in parallel on the cluster, you must submit your jobs to a work queue as instructed below. This is different than Lab1, so please read carefully:

Step 1:

Create your Hadoop Jar and transfer it to Hpc-class (done above!)

Step 2:
----
hadoop jar /home/<your id>/WordCount.jar WordCount /cpre419/shakespeare /user/<your id>/lab2/exp1/output/

Other information:

Make sure that the output location “/user/<your id>/lab2/exp1/output” are empty before running a new job, otherwise job will fail. There is one output file generated in the output location for every reducer. Show the output files to the lab instructor.

Experiments 2. (50 points)

A bigram is a contiguous sequence of two words within the same sentence. For instance, the text “This is a car. This car is fast.” Has the following bigrams: “this is”, “is a”, “a car”, “this car”, “car is”, “is fast”. Note that the two words within a bigram must be a part of the same sentence. For example, “car this” is not a bigram since these words are not a part of the same sentence. Also note that you need to convert all upper case letters to lower case first.

Task: Write a Hadoop program to identify 10 most frequently occurring bigrams, along with their frequencies of occurrence.

You can assume that each call to the Map method receives one line of the file as its input. Note that it is possible for a bigram to span two lines of input but they are in two mapper input splits; you can ignore such cases and you do not have to count such bigrams. Remove any punctuation marks such as “,”, “.”, “?”, “!” etc from the input text. It can be done in the mapper, regex:

```
String.replaceAll("[^a-z0-9]", "")
```

Question: Think about how you might be able to get around the fact that bigrams might span lines of input. Briefly describe how you might deal with that situation? (5 points)

Hint: In this task, you might need multiple rounds of MapReduce.

Think about the following in designing your algorithm:

- The number of rounds of map reduce
- What is the load on a single reducer

Make sure that the output path of each MapReduce job is a unique directory that does not conflict with the output path of another job. Use the following path to avoid conflict:

```
/user/<your id>/lab2/exp2/output
```

Make sure that the output location “/user/<your id>/lab2/exp2/output” and your temp directory “/user/<your id>/lab2/exp2/temp” are empty before running a new job, otherwise job will fail.

Skeleton code (Driver.java) is provided to help you to start with the exercise. The code is well commented so as to help you understand it. Read the comments in detail and try to understand the code. Then you can build on this code to write your program.

You can use the Shakespeare corpus as the testing input to your program. It is uploaded on the hdfs at the following location: /cpre419/shakespeare
Once your program runs successfully on this input, you can try to run your program on larger input files. You will find a much larger dataset in the location: /cpre419/gutenberg

Please submit the source code, in addition to the outputs for both the datasets: Shakespeare and Gutenberg. Your output must include the most frequent bigrams for each letter and their frequencies of occurrence.