CprE 419 Lab 2: Text Analysis using Hadoop MapReduce

Department of Electrical and Computer Engineering
Iowa State University
Spring 2013

Purpose

The goal is to introduce you to the MapReduce programming model under Hadoop. MapReduce is a significant abstraction for processing large data sets since it has been applied successfully to a number of tasks in the past, including web search. At the end of the 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 blackboard.

- A write-up answering questions for each experiment in the lab. For output, cut and paste results from your terminal and summarize when necessary.
- Commented Code for your program. Include all source files needed for compilation.

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 $k \subset K$, 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

Make sure you have the following:

- IP Address of the Hadoop Master Node
- Login id and private key file
- URL for the Hadoop Namenode
- URL for the Hadoop Tasktracker

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

You can write your Java program on your local system (using an IDE such as Eclipse). There are two ways you can compile a Hadoop Mapreduce program.

1. You can compile the program using Eclipse, while linking the Hadoop libraries, as described in Lab 1. Note that Hadoop only needs your jar file to run a MapReduce job, so you can generate the jar file locally and upload it to the server.

2. You can also compile and generate the jar files on the server as follows. You have to first upload the source files to the server. Then do the following:

   ```
   $ javac ---classpath $HADOOP_HOME/hadoop---0.20.2---core.jar WordCount.java
   <other files if necessary >
   $ jar ---cvf WordCount.jar *.class
   ```

The MapReduce program can then be started in the cloud as follows:

```
$ hadoop jar WordCount.jar WordCount /user/<User ID>/Lab2/input/   /user/<User
ID>/Lab2/output/
```

Notes

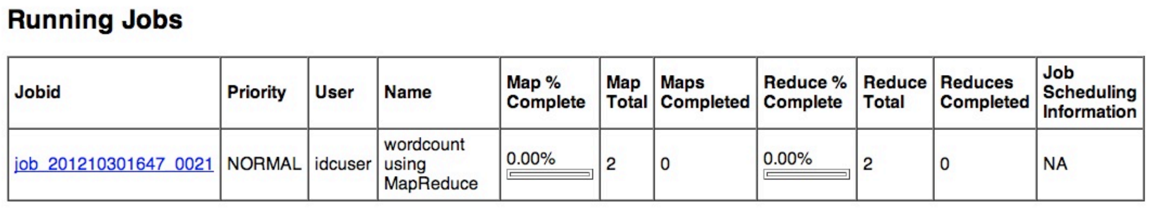
- The MapReduce programs read input from the HDFS and write their output to HDFS. However, you don't necessarily need to use the HDFS API while programming with MapReduce.

- 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.

- 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/r1.0.0/api/index.html

- Once you submit a Hadoop job, you can monitor it through the Tasktracker URL, whose output may look like this.



Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201210301647_0021	NORMAL	idcuser	wordcount using MapReduce	0.00%	2	0	0.00%	2	0	NA

Experiments

1. The WordCount program counts the number of occurrences of every distinct word in a document. Download the file WordCount.java and compile the program in the server as described above. Then run the program as follows:

   ```
   $ hadoop jar WordCount.jar WordCount /datasets/Lab2/Shakespeare  /users/<User ID>/Lab2/Exp1/output
   ```

   Make sure that the output location "/users/<User ID>/Lab2/Exp1/output" is empty as the job will fail otherwise. There will be one output file generated in the output location for every reducer. Show the output files to the lab instructor. (20 points)

2. 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 "car this" is not a bigram since these words are not a part of the same sentence. Also note that we have converted all upper case letters to lower case.

   Task: Write a Hadoop program to identify the 10 most frequently occurring bigrams in an input text corpus, 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; you can ignore such cases and you do not have to count such bigrams. Also remove any punctuation marks such as ",", ".", "?", etc from the input text (this can be done in the mapper), and convert all letters to lowercase.

   Hint: You might need multiple rounds of MapReduce.

   Think about the following in designing your algorithm:
   - The number of rounds of mapreduce
   - The communication within each round
   - 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: "/users/<User ID>/Lab2/Exp2/output".

   A skeleton code 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 input to your program. It is uploaded in the server at the following location: “/datasets/Lab2/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: “/datasets/Lab2/Gutenberg”.

Hint: For long running jobs call context.progress() in your code to avoid timeout dump.
(50 points)

Please submit the source code along with the outputs for both the datasets: Shakespeare and Gutenberg. Your output must include the 10 most frequent bigrams and their frequencies of occurrence.