1st Assignment

Due: 16 February 2017

A variation of an inverted index

You are asked to implement an inverted index in MapReduce for the document corpus of pg100.txt (from http://www.gutenberg.org/cache/epub/100/pg100.txt), pg31100.txt (from http://www.gutenberg.org/cache/epub/31100/pg31100.txt) and pg3200.txt (from http://www.gutenberg.org/cache/epub/3200/pg3200.txt). This corpus includes the complete works of William Shakespear, Mark Twain and Jane Austen, respectively.

An inverted index provides for each distinct word in a document corpus, the filenames that contain this word, along with some other information (e.g., count/position within each document). For example, assume you are given the following corpus, consisting of doc1.txt, doc2.txt and doc3.txt:

doc1.txt: "This is a very useful program. It is also quite easy."
doc2.txt: "This is my first MapReduce program."
doc3.txt: "Its result is an inverted index."

An inverted index would contain the following data (in random order):

this	doc1.txt, doc2.txt
is	doc1.txt, doc2.txt, doc3.txt
a	doc1.tx
program	doc1.txt, doc2.txt
…	

(a) (30) Run a MapReduce program to identify stop words (words with frequency > 4000) for the given document corpus. Store them in a single csv file on HDFS (stopwords.csv). You can edit the several parts of the reducers' output after the job finishes (with hdfs commands or with a text editor), in order to merge them as a single csv file.

i. (10) Use 10 reducers and do not use a combiner. Report the execution time.
ii. (10) Run the same program again, this time using a Combiner. Report the execution time. Is there any difference in the execution time, compared to the previous execution? Why?
iii. (5) Run the same program again, this time compressing the intermediate results of map (using any codec you wish). Report the execution time. Is there any difference in the execution, time compared to the previous execution? Why?
iv. (5) Run the same program again, this time using 50 reducers. Report the execution time. Is there any difference in the execution time, compared to the previous execution? Why?

(b) (30) Implement a simple inverted index for the given document corpus, as shown in the previous Table, skipping the words of stopwords.csv.

(c) (10) How many unique words exist in the document corpus (excluding stop words)? Which counter(s) reveal(s) this information? Define your own counter for the number of words appearing in a single document only. What is the value of this counter? Store the final value of this counter on a new file on HDFS.

(d) (30) Extend the inverted index of (b), in order to keep the frequency of each word for each document. The new output should be of the form:

this	doc1.txt#1, doc2.txt#1
is	doc1.txt#2, doc2.txt#1, doc3.txt#1
a	doc1.txt#1
program	doc1.txt#1, doc2.txt#1
…	

which means that the word frequency should follow a single '#' character, which should follow the filename, for each file that contains this word. You are required to use a Combiner.

BONUS: Relative Frequencies (30)

Compute the relative frequencies of each word that occurs in the three documents in the previous corpus and output the top 100 word pairs sorted by decreasing order of relative frequency.

The relative frequency (RF) of word B given word A is defined as follows:

$$f(B \mid A) = \frac{\text{count}(A,B)}{\text{count}(A)} = \frac{\text{count}(A,B)}{\sum_{B'} \text{count}(A,B')}$$

where count(A,B) is the number of times A and B co-occur in a line and count(A) the number of times A occurs with anything else. Intuitively, given a document corpus, the relative frequency captures the proportion of times the word B appears in the same line as A.

(a) Write a MapReduce program which uses the Stripes approach.

(b) Write a MapReduce program which uses the Pairs approach.

(c) Report and explain the relative performance of these two approaches.

TIPS: Ignore letter casing (e.g., use the Java String's method toLowerCase() for each input line, before you further process it). You can use the StringTokenizer class, or the split() method of Java Strings to tokenize an input string. Start your tests with the two smallest files as input and when your code works, include the third, biggest file. You should use only TextInputFormat and TextOutputFormat.