Rohan Patel

Salvator Avena

Dr. Chatree Sangpachatanaruk

CS 1699 Special Topics in Cloud Computing

Wednesday April 29, 2017

Hadoop vs Spark

We wrote this program using PySpark because of the ease of implementation. We were able to take our 400+ line hadoop and rewrite it around Spark with around 50 lines of code to generate the inverted index and around 100 lines of code for the UI. Spark also needs significantly less steps to set up properly as compared to hadoop so it took us less time to get our program up and running. We also had the benefit of our program running faster with the new implementation, especially the step for creating the inverted index which had a significant difference in runtime between spark and hadoop.

Spark is an open-source cluster-computing framework. Benefits of Spark include speed, wide library of functions, and the ability to work with different languages such as Scala, Java, and Python. By using spark, writing a wordcount program can be reduced to about three lines of code/three function calls (to flatMap, map, and reduce), as shown below:

‘‘‘text\_file.flatMap(lambda line: line.split())

.map(lambda word: (word, 1))

.reduceByKey(lambda a, b: a+b)’’’

Spark also allows programs to run up to 100 times faster Hadoop MapReduce in memory or up to 10 times faster on disk.

We implemented this version of tinyGoogle by first writing the code to generate the inverted index. We were able to do this by modifying wordcount to the version shown below:

‘‘‘counts = sc.wholeTextFiles(sys.argv[1], 1)\

.flatMap(lambda (name, content): map(lambda word: (word, name), content.split(' ')))\

.map(lambda (word, name): ((cleanWord(word), cleanFile (name)), 1))\  
.reduceByKey(lambda count1, count2: count1 + count2)\  
.map(lambda ((word, name), count): (word, name, count))’’’

This modified version of wordcount was able to grab the filename of each file in the directory that we passed in and then put it in the format of the term, filename, and count. We were then able to format this data in such a way to build the full inverted index by further reducing it by the term as the key and store the output in a JSON. The output JSON allowed us to be able to access the inverted index later instead of generating a new one every time we wanted to search for a term. An improvement we believe that we could have made on this would have been to add a another reduceByKey after the second .map call but we did not know if we would be able to take the output from that and easily convert it to a json.

Originally we wrote the program to run off the command line where you would enter “spark-submit file:///path/to/script file:///path/to/dir/\*.txt ‘term’” where you would specify the full path to the script, the full path to the directory with text files and the term(s) you want to search. After building a functioning inverted index and search feature we decided to build a UI around it.

We wrote our UI around saving the output JSON from the modified wordcount so that it could be quickly searched. The UI first prompts the if they want to build a new inverted index or use an existing one stored as a JSON. Afterwards it prompts the user to enter search terms. Each search term is then cleaned and turned to lowercase. After this it outputs each word and its frequency in each document and that word’s weight relative to each document. We wanted to use weights of each word to combine the outputs and print out the combined weight and frequency and weight of the words per document. Unfortunately we were unable to implement a working version of this so the program currently outputs each word being searched for individually.

In conclusion, if we compare Spark vs Hadoop MapReduce we found Spark to be the superior implementation. Spark had a much greater ease of implementation. In addition, the inverted index was generated in significantly less time than in the Hadoop implementation and the searches were significantly faster.