CS-839 NoSQL Systems

Dr. Vinu Venugopal, Divyam Agrawal, Aniket Mitra & Nandakishore S Menon

Assignment - II

Submit your solutions in a single zip file with your roll number as the file name on or before the due date (specified in the LMS). Add instructions for running the solutions to each problem in separate files named `problem_X.txt`*. Each group will have to demonstrate the submitted solutions to one of the teaching assistants. Each member in the group may be evaluated independently during the demo session. All the members in a group must participate in the demo.*

MapReduce & Apache Hadoop

Problem 1. Part-of-speech tagging using Pairs and Stripes — 9 Points

Each word in a sentence falls into one of several distinct grammatical categories, referred to as parts of speech. The process of assigning words in a sentence to their appropriate part of speech, such as a verb, noun, adjective, etc., is known as part-of-speech tagging, or POS tagging. Employ the MapReduce idea to fulfil the following requirements.

- From all words in the `Wikipedia-EN-20120601_ARTICLES.tar.gz` corpus, calculate the count of each POS tag (AUX, PRON, VERB, PRON, ADP, PROPN, PUNCT, NOUN, ...) across all the articles using a MapReduce job.
- Create two MapReduce programs: one that uses the Pairs approach and the other that uses the Stripes strategy. Compare the execution time of your programs.

Note: To identify a possible set of POS tags, you may use the OpenNLP library and a pre-trained POS tag model, which are available from `opennlp-tools-1.9.3.jar` and `opennlp-en-ud-ewt-pos-1.0-1.9.3.bin` on LMS. You may check the PosTag.java file on the LMS to learn more on how to use the OpenNLP library. Instructions for running PosTag.java and the required additional jar files are also provided. While running your hadoop job from the terminal, include the `opennlp-tools-1.9.3.jar` to the class path as shown here, `HADOOP_CLASSPATH` environment variable, as follows: `export HADOOP_CLASSPATH="<PATH_TO_LIBRARIES>/*"`

Problem 2. Indexing Documents via Hadoop — 12 Points

1. Implement a pair of a `Map` and a `Reduce` function which, for each distinct term that occurs in any of the text documents in `Wikipedia-EN-20120601_ARTICLES.tar.gz`, counts the number of distinct documents in which the term appears. We will call this value the Document Frequency (DF) of that term in the entire set of Wikipedia articles. Store the resulting DF values of all terms in a single TSV file with the following schema:

   ```
   TERM<tab>DF
   ```

   While generating the output in the above format, consider filtering out all terms that belong to the stopwords.txt file shared on LMS. (You may perform this filter operation in your map method.) Identify the top 100 terms with a high document frequency. Use those terms alone for the next sub problem.

   Hint: Also here consider the Porter stemmer that is available from `opennlp-tools-1.9.3.jar` *on LMS. After importing the library with* `import opennlp.tools.stemmer.PorterStemmer;` *and creating an instance of* `PorterStemmer stemmer = new PorterStemmer();` *use* `stemmer.stem(token);` *for stemming an input token.* **6 Points**

2. Implement another pair of a `Map` and a `Reduce` function which, for each document in `WikipediaEN-20120601_ARTICLES.tar.gz`, first counts the number of occurrences of each distinct term within the given document. We will call this value the Term Frequency (TF) of that term in the given document. Implement a stripes algorithm here as we are dealing with just 100 terms.

   In a second step, your combined MapReduce function should multiply the TF value of each such term with the inverse of the logarithm of the normalized DF value calculated by the previous MapReduce function, i.e., SCORE = TF$\times$log(10000/DF+1) for each combination of a term and a document. You may cache the former TSV file with the DF values by adding it via `Job.addCacheFile(<path-to-DF-file>)` in the driver function of your MapReduce program. The `Map` class should then load this file upon initialization into an appropriate main-memory data structure.

   The result of this MapReduce function should be a single TSV file that has the same schema as the file we used in the previous exercise sheets: **6 Points**

   ```
   ID<tab>TERM<tab>SCORE
   ```

   Hint: You may use either the given document URL's or simple integer id's for the `ID` *field of this TSV file (as long as they uniquely identify the original text article).*