Applications of MapReduce

CS 6350 - Big Data Management & Analytics

Assignment 2

Instructions

- This assignment is based on applications of MapReduce. You are free to do this in either Scala (preferred) or Java. For Scala, the code can be written either on Databricks or on UTD cluster.
- Please use parameters and not hard coded paths. All instructions for compiling and running your code must be placed in the README file.
- You have to use MapReduce as the programming framework. Other solutions will not be accepted.
- All work submitted must be your own. Do not copy from online sources. If you use any references, please list them.
- You should use a cover sheet, which can be downloaded from: http://www.utdallas.edu/~axn112530/cs6350/CS6350_CoverPage.docx
- You are allowed to work in pairs i.e. a group of two students is allowed. Please write the names of the group members on the cover page.
- **You have a total of 4 free late days for the entire semester. You can use at most 2 days for any one assignment. After four days have been used up, there will be a penalty of 10% for each late day. The submission for this assignment will be closed 2 days after the due date.**
- Please ask all questions on Piazza, not via email.

PageRank Calculation

1. We discussed PageRank(PR) creation using MapReduce in class. In this part of the assignment, you will write code that calculates PR for a set of pages from the Apache Foundation that have been crawled by a search engine. The dataset can be downloaded from: `http://www.utdallas.edu/~axn112530/cs6350/data/PRData/webcrawl`
   The schema of the dataset, with fields being tab separated, is as follows:
   First Field: URL
   Second Field: Starting PageRank (1.0 for every page)
   Third Field: List of links found at that URL

   You have to read in the file to either a plain RDD or Dataframe and run the PageRank algorithm to compute the rank of every page. To make things simple, you can make the following assumptions:

   - The PR is computed only on the basis of neighboring inlinks / outlinks and not due to random surfer factor. So, the PR for a node x having inlinks from pages $t_1, t_2, \ldots, t_N$ and having out-degree i.e. number of outlinks $C(t)$ as:

     $$PR(x) = \sum_{i=1}^{N} \frac{PR(t_i)}{C(t_i)}$$

     Note: You have to compute PR only up to 3 decimal points.
   - The terminating condition of the algorithm will be either if the ranks converge or until a maximum of 100 iterations. That is, you can set a maximum number of iterations to be 100.
   - You can use any numerical processing library, but you can't use any library that computes PageRank.

   Your program should have a parameter for the location of the input file. You have to output the PR of each page in the list sorted in descending order by the PR values.

Inverted Index Creation

2. In the class, we discussed how an inverted index can be created using MapReduce. In this part of the assignment, we will use Spark to construct inverted index on a text corpus consisting of data from 20 newsgroups. This dataset is available at: `http://qwone.com/~jason/20Newsgroups/`. It is preferred that you use the version that duplicates and headers removed and can be downloaded from: `http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz`. The dataset has email conversations about 20 different topics e.g. "sci.crypt", "comp.sys.ibm.pc.hardware", "comp.os.ms-windows.misc". You have to use the **train** part of the dataset for building **tf-idf** data structure.

   Below are the requirements and assumptions:

   - You can combine all files for each category, and call it one **document**. For example in the directory "sci.electronics", there are 591 different files that you can combine together to call it the **sci.electronics document**. You will thus end up with 20 large documents created from the training part of the dataset.
   - You have to build a tf-idf index for **each term having length greater than 5**. We discussed tf-idf in class, and it states that the weight of term i in document j is:

     $$w_{\text{tf-idf}} = tf_{i,j} \times log\frac{N}{n_i}$$

     where:
     $tf_{i,j}$ measures the frequency of term i in document j
     N is the total number of documents in the collection, which is 20 in this case
     n_i refers to the number of documents that contain the term i
   - You are free to use any text processing framework, but can not use any library that creates inverted index

   Your code will be tested by giving a list of words in a file and it should output the tf-idf values for the top 5 documents for that term in descending order of tf-idf values. An example of such a file can be seen at: `http://www.utdallas.edu/~axn112530/cs6350/data/testInvIndex.txt`

   Your code should have parameters for the input path of the main training directory i.e. "20news-bydate-train" and a parameter for the test file.