Class 2 Homework

Realtime and Big Data Analytics

Summer 2017

Homework

Class 2

1. If you have not already done so, please install the Cloudera Quickstart VM, or install Hadoop in pseudo-distributed mode and run the MaxTemperature program.
   Follow instructions from Class 1 assignment and please hand in on NYU Classes.

   For help, send an email addressed to me and our TAs.

2. Please complete last week's TDG reading if you haven't already done so.

3. Please read:
   - Chapter 2: Page 30-37 (Scaling Out until Streaming)
   - Chapter 3: Stop at top of p.48 (HDFS Federation), read p. 70-71 (Network Topology and Hadoop), read bottom of p.73-top of p.76 (Replica Placement until Parallel Copying with distcp).

4. Please read: "The Google File System", by Ghemawat, Gobioff, and Leung.
   Link: http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf
   Read sections 1 and 2 at least. You will notice a difference in terminology when compared with HDFS.
   Please summarize the paper in one paragraph.

Homework

Class 2

---

5. This is a WordCount-based problem - the goal is to find the *number of lines* containing a specific search term.
Write a Java (or Python or C/C++) program that:

a. Searches for all of the following strings in the input file containing tweets (you can provide the search terms as parameters, or hardcode them): **`hackathon, Dec, Chicago, Java`**

b. Accepts a small input file to be searched containing lines of the form: ***Date,Time,Name,Tweet***
Here is the ***exact data*** to type into your input file:

```
09-Dec-16,6:00PM;#Hackatopia,Tribeca Film Hackathon: Code As A New Language For Content Creators Hackathon
28-Oct-16,7:00PM;#NYCHadoop,Hadoop-NYC Strata/Hadoop World Meetup at Google NYC
31-Dec-16,3:00PM;#Hackatopia,Designers, Developers, Doers, don't miss this upcoming Chicago hackathon
```

c. Your code will search for all of the search strings in the input file and output the ***number of tweets*** that contained each search string (not the number of occurrences of a search string). The matching is **not case sensitive**, i.e. if searching for the search string ***`hackathon`***, all of the following are a match: **`hackathon, Hackathon, hACKathon`** (and any other combination of upper and lower case characters).

d. Your code should output the number of tweets that contained each search string. Using the input data above, the resulting counts will be:

```
Chicago 1
Dec 2
Java 0
hackathon 2
```

e. Upload homework to NYU Classes. To receive full credit, please hand in all of the following items on or before the due date:
- Your program, your input file, and job output.
- Evidence that the program ran successfully (e.g. screen shots or output log as you did for homework #1)

Homework

Class 2

6. This is a similar program to the one you just wrote in the previous step, but this time, you will use the ***MapReduce framework*** and develop an algorithm based on WordCount that takes advantage of the cluster resources. This algorithm is very different from your previous one - it needs to split the algorithm into work that the map phase can do, and work that the reduce phase can do.

Your program:

a. Searches for all of the following strings in the input file containing tweets (you can provide the search terms as parameters, or hardcode them): **`hackathon, Dec, Chicago, Java`**

b. Accepts the ***same small input file*** you used in the previous exercise and searches it for the search strings.

c. The Mapper code will search the input file line by line to find matches. The matching is not case sensitive (same as before).

The mapper code should not do any summing or buffering. The summing must happen in the reducer code.

d. You are required to use a Reducer. The Reducer code will input the key-value pairs generated by the map phase and output the number of tweets that contained each search string. Using the input file, the resulting counts will be:

```
Chicago 1
Dec 2
Java 0
hackathon 2
```

e. Upload homework to NYU Classes. To receive full credit, please hand in all of the following items:

- Your program, your input file, and job output.
- Evidence that the program ran successfully (e.g. screen shots and output log as you did for homework #1)

Note: Your plain old Java algorithm (from the previous exercise) is not what you want to use in this MapReduce solution. Think about the example covered on the board in class where temperatures (values) were sorted by year (the key). These key-value pairs are guaranteed to arrive at the reducer(s) sorted by ***key***. The reducer can iterate through the ***values*** associated with a given key and process them. In the MaxTemperature example, that *processing* was to select the max temperature by iterating through the values. Think about what a reducer should do for a WordCount algorithm to be efficient in a distributed system - write your algorithm so the summing happens in the reducer.