TWEETWORDCOUNT ARCHITECTURE

RANI FIELDS

W205-2, Exercise 2

AUGUST 16, 2017

TABLE OF CONTENTS

1. WHY APACHE STORM FOR TWEET PROCESSING?

When working with streaming text data, two of the predominant questions come to mind: How do I properly parse my text efficiently and effectively while still maintaining throughput? While many solutions exist, we do have one solution of note: Apache Storm. The beauty of Apache Storm for parsing and storing Twitter tweets is multi-fold: not only do we have that aspect of efficiently and effectiveness (provided your code is properly designed), we also have the added benefit of a highly modular and quasi-scalable architecture via parallelism.

Despite its benefits, implementing a Twitter stream in Apache Storm is not without its challenges. Of note, if you are looking for a truly streaming solution, not a piece-wise one where data is only handled in bursts, two major issues might cause substation challenges. First, the targeted Twitter suite for this course, Tweepy, must run on its own thread. Second, due to a bug stemming from Python2.x libraries, you will need to have the ability to restart Tweepy streams on-demand. Please note that I have implemented both features into my code.

Once your Tweepy package problems have been addressed, there is one extra area of worry: Zookeeper. Specifically, after roughly 30 minutes (exact timings are not available), you enter a situation where Zookeeper could fail in several ways. While I have managed to address Tweepy-related errors, I have yet to find a solution for the Zookeeper-related failures.

2. Application Idea

I've implemented this application such that we have three primary layers: input, translate/parse, output. The first layer, the input layer, is defined its interaction directly with Twitter to collect tweets. The second layer, the translate/parse layer, is meant to take an existing tweet, and turn it into functional, useful words via the removal of stop words, splitting certain contractions, etc. The third layer is where we push our results to PostgreSQL.

Please note that PostgreSQL is of special importance to this project. Because many of our scripts revolve around Apache Storm and since the existing Apache Storm set-up does have some stability issues, we need to ensure that certain state data (word count data, specifically) are stored in our database. As such, I've taken special care to ensure the output layer is to some degree optimized. For instance, we try to maintain connections for as long as possible. We also attempt to minimize the number of calls made to PostgreSQL.

3. Directory and File Structure

The program directory is structured as such:

1. Base directory
    - a. File: architecture.pdf
    - b. File: readme.txt
    - c. File: plot.png
    - d. File: variables_dummy.rc
    - e. Folder: screenshots
    - f. Folder: installation
        - i. File: create_table.sql
        - ii. File: create_database.sql
        - iii. File: create_table.sh
        - iv. File: create_table.sql
        - v. File: requirements.txt
        - vi. File: readme.txt
    - g. Folder: scripts
        - i. File: finalresults.py
        - ii. File: histogram.py
        - iii. File: readme.txt
    - h. Folder: extweetwordcount
        - i. Spout path: src/spout/tweets.py
        - ii. Bolt path: src/bolts/parse.py
        - iii. Bolt path: src/bolts/tweetcounter.py
        - iv. Topology path: topologies/extweetwordcount.clj
        - v. Various other Apache Storm folders and files are present. These are unedited.

4. Provided Scripts

I've provided two scripts under the `scripts/` folder. We have finalresults.py and histogram.py. In the scripts folder, I've also included a readme file.

finalresults.py allows you to query the number of occurrences of some word. If you do not provide a launch parameter, it instead prints all words, sorted alphabetically.

histogram.py takes in some <min>,<max> launch parameter and outputs a list of all words whose count falls between the min and max values.

5. Requirements and Installation

I've included an `installation/` folder for installing all required python packages. In order to install the appropriate packages, please run `pip install -r requirements.txt`.

You'll also find a folder for creating the SQL table and database. Please use the database script before the table script. I've also included a file which, assuming you are using trust authentication, will automatically set up the database and table for you. Note that this will not drop and tables or databases.

6. Running TweetWordCount & Related Scripts

First, you will need to load your configurations. For this, you will need your database host, database user, and, if required, database password. You will also need your four Twitter credentials. Simply take the `variables_dummy.rc` file located in the base folder, modify it as required, and source the file (e.g. "`source variables_rani.rc`").

From here, simply change your directory to `extweetwordcount/` and run the command `sparse run`. You should not need any special launch parameters such as timeout parameters- I've taken special steps to prevent the need for any such parameters.

7. THOUGHTS FOR THE FUTURE

If we were to develop this system further, I would like to provide a few suggestions beyond simple bug fixes, all of which revolving around parsing our tweets for valid words:

1. A superior word merging system
    a. Fix likely misspellings via hash tables
    b. Merge similar words (e.g. 'co-worker' vs. 'coworker')
    c. Merge conjugated verbs into their infinitive form
2. Some form of syllable/letter duplicate removal system
    a. Will change words like 'hahahaha' into 'haha.'
    b. Will change words like 'loooove' into 'loove', or better, 'love.'
3. More exhaustive contractions handling systems
4. Allow negations and other similar concepts to be treated as a single word
    a. "good" and "not good" have two very different connotations
    b. This would be very useful for sentiment analysis