W205 Exercise 2 Report

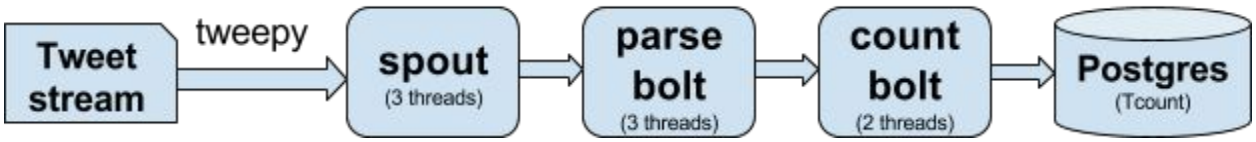
Lei Yang - 12/12/2015

Directory and file structure

- **setup/create_db.py** - script to create tcount database in Postgres
- EX2Tweetwordcount: streamparse project for the application
    - **topologies/tweetwordcount.clj** - Storm topology for the streaming analysis
    - **src/spouts/tweets.py** - script for tweet spout, using tweepy via Twitter API
    - **scr/bolts/parse.py** - script for parsing tweet, to extract words and filter out special characters and words of no interests
    - **scr/bolts/wordcount.py** - script to count words, and to store result in Postgres database.
- serving scripts
    - **finalresults.py** - script to check count for all words or a specific word
    - **histogram.py** - script to show the words with counts in the specified range
- screenshots - a few png to show the work
- sample code - code snippets for using psycopg2 and tweepy.

Application overview

Data flow: based on the topology defined in tweetwordcount.clj



Description:

- **spout** - using tweepy, via Twitter API, we stream any tweets that contains any of these characters: ‘a’, ‘the’, ‘i’, ‘you’, ‘u’. Since these are basic elements of English sentence, we expect to obtain the majority of the tweets with this filter. With 3 threads, and an 100-sized queue, we have buffer of 300 tweets at any given time. All tweets are emitted to the parse bolt for further processing.
- **parse bolt** - in this process, we extract words by splitting the tweets. And each word goes through the following steps:
    - get rid of any number, and any word connects with a number (e.g. tom123). This is done by regular expression filter:
        - `re.sub(r'\w*\d\w*', '', tweet).strip()`
    - convert everything to lowercase

    - get rid of words that start with # (hashtag), @ (mention), rt (retweet), and http (URL).
    - strip leading and trailing special characters:
        - `word.strip("\"?><,\'.:;!~[]()&%$*/=+-_^$\\")`
    - we define an array (no_count) that holds words that we won't count, these are the words appears most frequently in a sentence and yet contain little information of interests. Furthermore, words for 1 or 2 characters won't be counted either.
        - note: other processing can be done to further reduce noisy words like 'you're', 'i'll', 'there' etc. And maybe only keeping nouns and adjectives would be better for the purpose, but we didn't spend more effort on this.
    - after going through the above steps, the word is emitted to next count bolt.
- **count bolt** - we have 2 bolt threads, and we apply word grouping rule here between threads parse and count bolts.This will *guarantee* same word always goes to the same count thread, such that the count is unique and accurate.
    - as each count thread initializes, it will sync with the database first. The bolt will buffer all existing/previous word counts and continue with the incoming words. The reason here is to reduce the communication frequency with the database, as the program doesn't need to always check if a word is new or not.
    - for every new count, the process will update record accordingly in the database.
        - note: for real use case, the exact count is probably not super important, thus instead of updating every single value, we can update it for every 10 counts. This will essentially reduce database iops load 10 times. We didn't implement this strategy in this exercise tho, but it fairly straightforward.
    - to avoid log flooding and to be eye-friendly, we only log the word for every 100 counts.
- **postgres** - a *tcount* database is created, and a table *tweetwordcount* is used to record the counts. The table has two columns: word (text) and count (int), and a primary key is defined on word to avoid duplicate word entries.

Deployment

- system requirements:
    - python 2.7 with packages: streamparse, tweepy, psycopg2
    - Apache Storm, Postgres installed and properly configured
- steps
    - execute: `python create_db.py` in setup dir to create postgres table
    - apply a Twitter API credential: instructions here. And put the credentials in src/spouts/tweets.py
    - execute: `sparse run`, under EX2Tweetwordcount folder (press enter if you are root), Storm will start and eventually word count will start populating in the log:
        - example: *29052 [Thread-41] INFO backtype.storm.task.ShellBolt - ShellLog pid:4680, name:count-bolt weather: 100*

Results

- a few screenshots to validate the whole thing is working:
  - **screenshot-storm-startup.png**: process of streamparse creating spout and bolts.
  - **screenshot-streaming-log.png**: log word for every 100 counts, to avoid log flooding and be eye-friendly.
  - **screenshot-top-20-count-and-total-word.png**: it's good to see 'love' and 'like' are top 2 words. More than 60k words, a lot noise, need to do a better parsing job.

- run two serving scripts to check results:
  - `python finalresults.py google`
    - *Total number of occurrences of "google": 224*
  - `python finalresults.py`
    - there is an interesting "anti-word-counting" tweet: (win+a+two-night+ritz-carlton+experience.+sign+up+using+my+link+and+i+will+receive+an+additional+chance+to+win, 2)
    - a better parsing would be to replace all special characters with spaces (using re) before split the tweet. This will significantly reduce the number of words.
  - `python histogram.py 6000 7000`
    - *follow: 6851 make: 6611 best: 6571 why: 6520 would: 6478 back: 6447 got: 6413*
    - *channel: 6402 never: 6344 here: 6164 think: 6115 black: 6059 updates: 6050*