MIDS W205 Exercise 2

Alan Wang

Introduction

In this exercise, I explored a streaming application that analyzes the Twitter data. In order to explore complex implementation, I wrote codes by using an existing code base. I used Streamparse as seen in Lab 9 with a recommended topology. The application reads the stream of tweets from the Twitter streaming API, track and parses them, counts the number of each word in the stream of tweets, and writes the final results back into a Postgres database. I also developed three utility functions to track and demonstrate the result from the Postgres database.

Main Components

To accomplish the goal of streaming Twitter data, collect and maintain the running words counts in a Postgres database, I divide the task into the following components

- Sign in and connect to tweets.
- Streaming processing the incoming tweets
- Parse, screen and count the incoming words from the tweets
- Collect and store the word counts into a relational database

Technologies used in this exercise:

Apache Storm, Amazon EC2, python, Twitter API, Streamparse, Postgres, and Psycopg

Overall Architecture

Figure 1 shows the overall architecture of the application. Figure 1 also shows the storm topology that I used to develop as part of the application. Using Tweepy library, the application reads the live stream of tweets from twitter in the **Tweet-spout** component. The **Parse-tweet-bolt** parses the tweets, extracts the words from each parsed tweet and emits the words to the next bolt component (i.e **Countbolt**) in the topology**. Count-bolt** counts the number of each word in the received tuples and updates the counts associated with each words in the **tweetwordcount** table inside the **Tcount** database. **Tcount** is a postgres database.

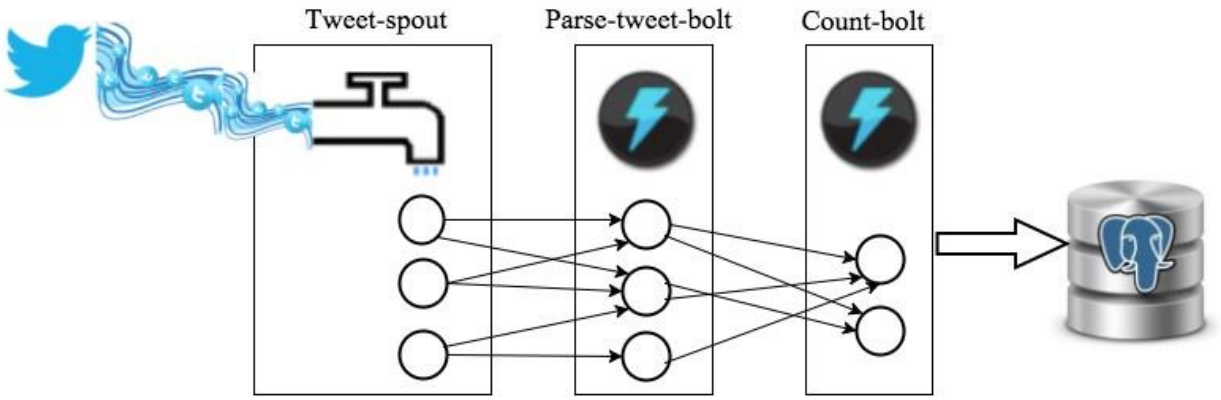

Figure 1: Application Topology

Storm Topology

The storm topology in Figure 1 can be precisely prescribed sparse clj code.

```
(ns tweetwordcount
  (:use     [streamparse.specs])
  (:gen-class))

(defn tweetwordcount [options]
   [
    ;; spout configuration
    {"tweet-spout" (python-spout-spec
          options
          "spouts.tweets.Tweets"
          ["tweet"]
          :p 1
          )
    }
                                        ;; bolt configuration
                                        {"parse-tweet-bolt" (python-bolt-spec
                                              options
                                              {"tweet-spout" :shuffle}
                                              "bolts.parse.ParseTweet"
                                              ["word"]
                                              :p 2
                                              )
                                         "count-bolt" (python-bolt-spec
                                              options
                                              {"parse-tweet-bolt" ["word"]}
                                              "bolts.wordcount.WordCounter"
                                              ["word" "count"]
                                              :p 2
                                              )
                                        }
                                    ]
                                )
```

The Tweet-Spout

In particular, I used one Spout and two Bolts. The Spout was based on Tweepy whose role is to connect to Twitter ad stream in related tweets for the specific ‘application’ I signed up Twitter for. At the spout stage, all message that contains words from a special predefined ‘interest group’ will be emitted to the process bolt. The interest group in the demon was default to be [‘a’, ‘the’, ‘I’, ‘u’, ‘you’] some of the most often used words in tweets to maximized the involved total number of

tweets emitted to show our processing capability. Please note at the Spout level, we were only specifying high level screening, deciding whether any incoming tweet contains keywords we are interested. Other than that, no word or character level parsing or processing is involved. We leave that to the two bolts.

The Parse-Bolt

The first bolt ParseBolt is used is going to parse the words from the tweets and splitting the tweets into words.. Moreover, some general pre-processing at the character level also take place here. For example, in this case, several meta character (e.g. hashtag) and punctuation marks are taken out. I could also have done some overall normalization like put all words into lower case or apply regular expression to make further transformations (like taking out the 's' at the end for plural words). But in the current implementation, we just weeded out a few more punctuations.

At the end, each tweets passing down to the ParseBolt is parsed and all words of the tweet is emitted to the second bolt.

The WordCount-Bolt

The second bolt, wordCount bolt, focus on one thing only: preserving the word counts.
and keep a record of the accumulated counts (of the overall appearances) in the Postgres database Tcount, at tweetwodcount table.

Specifically, for each word coming to the bolt processing, we will check the database, if there is already a record (word count) for that word, we will update the record (by adding 1 to the count). If there is none, then we will insert that word into the database by initializing its count to 1.

*One minor point, psycopg2's connect function does NOT create a new Database if the one specified does not exit. Instead an error message will be issued.

Utility Functions

In this step, your task is to develop two simple scripts that query the database and return specific results as follows:

1. `finalresults.py <word>`
   This script gets <word> as an argument and returns the total number of word occurrences in the stream. If run finalresults.py without an argument it will return all the words in the stream and their total count of occurrences, sorted alphabetically in an ascending order,

2. `histogram.py`
   This script gets two integers k1,k2 and returns all the words that their total number of occurrences in the stream is more or equal than k1 and less or equal than k2.

3. top20.py

I wrote a psycogs2 py script to retrieve top 20 words for one of the assignment in the project.

Note: This is the readme.txt I used in the submission directory, for your reference.

** All storm application AND the three utilities are stored under the same directory as instructed at EX2Tweetwordcount.
I will refer this directory as the 'base' directory. This might be the closest to 'turn-key' installation.
*** Assuming python, sparse, and Postgres are all installed as the document has indicated.

Step 1: make sure the database Tcount is properly created and the table tweetwordcount is defined.
Since the Psycog2 connect function turns out does NOT automatically create new databases, we need to initialize
and create the Tcount database.
a. I created a sub-directory under the base. It is named 'db-init'. Please cd to that directory.
b. If there was no previous database existing as Tcount, then please do $./createdb.sh
c. If there is already a previous database Tcount, then please do $./dropdb.sh to delete it first, then do 1.b
d. Either way, afterwards, please perform $python createtable.py to create the database.

Step 2: once we are sure the Tccount database is properly initialized, we can move on to storm
At the Ex2Tweetwordcount directory, you should be able to run $sparse run directly. and you should see the outcome
rolling out after the second time the "Enter" key is hit.

Step 3: You can let Step-2 running for a while, preferably from a different putty, you should be able to check out the following utilities developed
as instructed.

a. finalresults.py: this can be performed with one argument or without arguments. You should see the former will yield
one line telling you the word count for the specified word (including 0). In the non-argument version, all words
in the database will be displayed, which could be annoying if after a long execution of the system.
b. histogram.py: This was developed with a board range of tolerance of input conventions. Basically it will take
all argument which can be resolved to <k1>,<k2> where k1 and k2 are integers and k1 <= k2. (You can insert blan around ',')
c. If interested, you can also run python top20.py to get the top 20 wordc ounts .