Jasen Jones
w205 Exercise 2
December 13, 2015

Final Project Architecture

Summary

For Exercise 2 I've built an end-to-end application that streams tweets and outputs various summaries of the words contained in those tweets. The application is built upon Apache Storm and Python, and modeled largely after Lab 9's Streamparse structure but uses Postgres database management system. The python scripts 'finalresults.py' and 'histogram.py' allow users to interact with the database to retrieve the count of a word (or set of words), the number of words with counts between a specified range, and an alphabetized list of words with their counts. Assuming users are running the ucbw205_complete_plus_postgres_PY2.7 AMI and that "psycopg2" is already installed on the user's instance, all files that the user will need to run the application are included my Github repository.

Database Structure and Setup

This exercise uses Postgres for database management, requiring the user to complete a few setup steps before running the application. These steps and any necessary command line code are outlined in readme.txt in the main folder of my Github repository for this project. The application assumes that the user is logged in as 'w205', so it's important to switch to that user before cloning the github repository.

The instructions in readme.txt set up a new database within Postgres called "tcount" - note that while the instructions for this exercise always capitalize "tcount" I have not done so in my project. Postgres is the initial user of this database, so some small permissions and ownership tweaking is necessary to ensure the 'w205' user can make edits to the "tcount" database using

PSQL. Once this is set up, the 'w205' user can then access the database with PSQL and create our table, "tweetwordcount" (note that this is also all lower-case).

With those two steps complete, our database and table are properly configured to run the application.

Storm Architecture

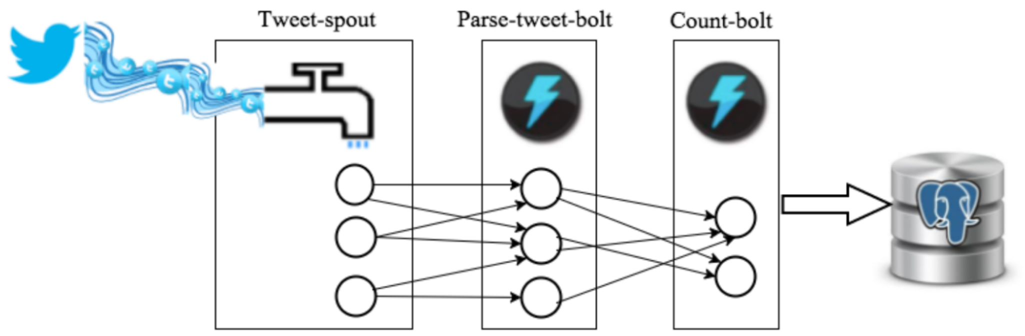


Figure 1: Application Topology

Our Storm application has three major components: a spout, a parsing bolt, and a counting bolt. The spout is split into two processes and streams tweet data using the 'tweepy' API. I tried splitting this into three processes but ended up getting many more "empty queue exceptions" which seemed to slow down the application when compared to two running processes. The spout takes the entire tweet (including metadata) and sends it to the parsing bolt.

The parsing bolt takes all the tweet data and extracts just the tweet text, then splits the text into individual words and adds them to a list which is then emitted to the counting bolt. My topology uses three parsing bolt processes as this seemed to be the sweet spot for speed in processing tweets without encountering an exorbitant number of empty queue exceptions. Each word has a set number of special characters removed and ensures that only ascii characters to appended to the list provided to the next bolt.

```
;; spout configuration
{"tweet-spout" (python-spout-spec
      options
      "spouts.tweets.Tweets"
      ["tweet"]
      :p 2
      )
}
;; bolt configuration
{"parse-tweet-bolt" (python-bolt-spec
      options
      {"tweet-spout" :shuffle}
      "bolts.parse.ParseTweet"
      ["word"]
      :p 3
      )
 "count-bolt" (python-bolt-spec
      options
      {"parse-tweet-bolt" ["word"]}
      "bolts.wordcount.WordCounter"
      ["word" "count"]
      :p 2
      )
```

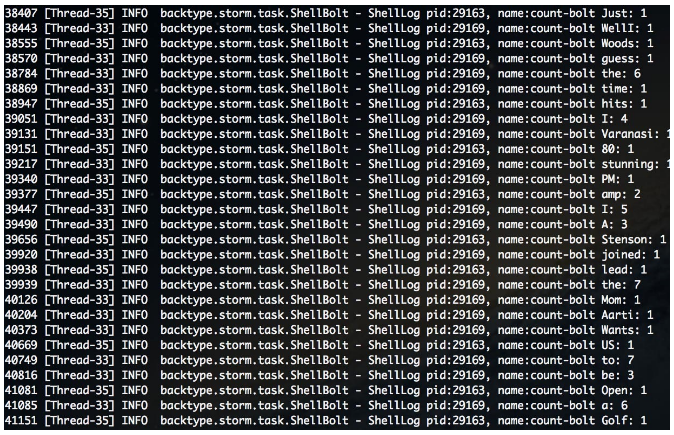
The counting bolt runs two processes and communicates the words provided by the parsing bolt to the "tweetwordcount" table in the "tcount" database with psycopg2. Psycopg2 connects to the "tcount" database as user 'w205' without a password and uses a cursor to manipulate the "tweetwordcount" table. This is why it's important that the user be logged in as 'w205' - other users will not have the correct permissions to manipulate the database. Each new word is inserted into "tweetwordcount" with a count of zero, which is immediately increased by one. Words that already exist in the table simply add one to their existing count. As an example, if the word "apple" appeared in "tweetwordcount" with a count of 15, the counting bolt would iterate once to make the count 16. If the word "awesome" does not appear in "tweetwordcount" then the counting bolt will insert that word into the table and set it's count to 0 before immediately adding one to that word's count.

Running the Script

Once the database and table are properly configured using readme.txt, the application may be run using the following command:

```
$sparse run
```

This will load the existing topology to begin streaming and counting tweets and interact with the "tcount" database. Each word will appear in the console with its local count, which should match the count seen in the "tweetwordcount" table if using a fresh database. If you already have data in the "tweetwordcount" table (from previous runs, for example), then the number shown next to each word is what was added to the existing count for each word.



```
38407 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt Just: 1
38443 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt WellI: 1
38555 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt Woods: 1
38570 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt guess: 1
38784 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt the: 6
38869 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt time: 1
38947 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt hits: 1
39051 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt I: 4
39131 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt Varanasi:
39151 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt 80: 1
39217 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt stunning:
39340 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt PM: 1
39377 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt amp: 2
39447 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt I: 5
39490 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt A: 3
39656 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt Stenson: 1
39920 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt joined: 1
39938 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt lead: 1
39939 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt the: 7
40126 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt Mom: 1
40204 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt Aarti: 1
40373 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt Wants: 1
40669 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt US: 1
40749 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt to: 7
40816 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt be: 3
41081 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt Open: 1
41085 [Thread-33] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29169, name:count-bolt a: 6
41151 [Thread-35] INFO  backtype.storm.task.ShellBolt - ShellLog pid:29163, name:count-bolt Golf: 1
```

If there's not already data in "tweetwordcount" then the application should be run for at least a few minutes to get interesting results for analysis. Use Ctrl-C to stop the streaming.

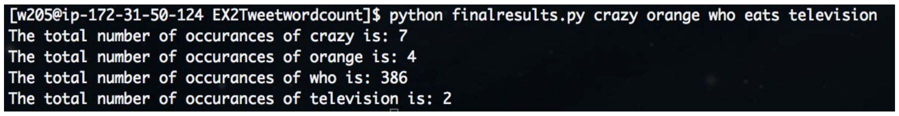
Analysis Scripts

I created two scripts that can be used to perform basic analysis of the data contained in "tweetwordcount". The file finalresults.py allows a user to view counts for one or more specific words or print an alphabetized list of all words that appear in the table and their respective counts. There are two different ways of using this script - the first is simply to use:

$python finalresults.py

This will return the sorted results of all words found in “tweetwordcount”. The second method is to add words to the end of your script call, like so:

$python finalresults.py orange apple



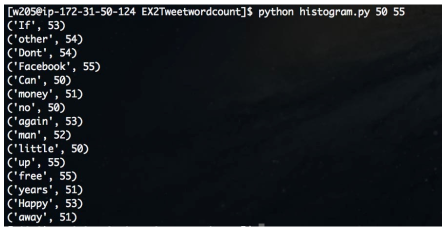
```
[w205@ip-172-31-50-124 EX2Tweetwordcount]$ python finalresults.py crazy orange who eats television
The total number of occurances of crazy is: 7
The total number of occurances of orange is: 4
The total number of occurances of who is: 386
The total number of occurances of television is: 2
```

You can add as many words as you want after “finalresults.py” - the script will return the word and the count for every word you list, assuming the word appears in the database. If nothing is printed then the word doesn't appear anywhere in our data.

The second script executes a simple query that returns all words that have counts within a user specified range. The user can run the script with a command like the following:

$python histogram.py 50 55

This will return all words whose total count falls within 50 and 55, inclusive.



```
[w205@ip-172-31-50-124 EX2Tweetwordcount]$ python histogram.py 50 55
('If', 53)
('other', 54)
('Dont', 54)
('Facebook', 55)
('Can', 50)
('money', 51)
('no', 50)
('again', 53)
('man', 52)
('little', 50)
('up', 55)
('free', 55)
('years', 51)
('Happy', 53)
('away', 51)
```

Conclusion

This exercise combines three major components: a Postgres database, a Storm application, and python scripts to summarize the data found in our database. Our application streams tweets using Tweepy, parses those tweets, then adds each word to our database and counts the number of times that word has been seen. Users can interact with the database directly or use the python summary scripts to retrieve certain types of information, like an alphabetized list of words and their respective counts, the count for user-specified words, or a list of words that have counts falling between a user-specified range. To start the application, refer to the readme.txt file found in the main section of the Github repository I've shared (username FrozenMangos).