Chris Sanchez
UC Berkeley MIDS
W205: Data Storage and Retrieval
Exercise 2

Exercise 2: System Architecture

Application Idea

This version of the Twitter Word Count application pulls live Tweets from the Twitter "garden hose", tokenizes the Tweets into single words, increments the count of each word as it passes through the pipeline, and then stores each word and it's updated count (in real-time) in a PostgreSQL Database. Two simple python scripts accompany the application that will return number of occurrences of a given word or provide a list of words within a given integer range as provided by user arguments (the same results can be achieved through the postgres command line, but the scripts allow a user not familiar with SQL syntax to query the Twitter results).

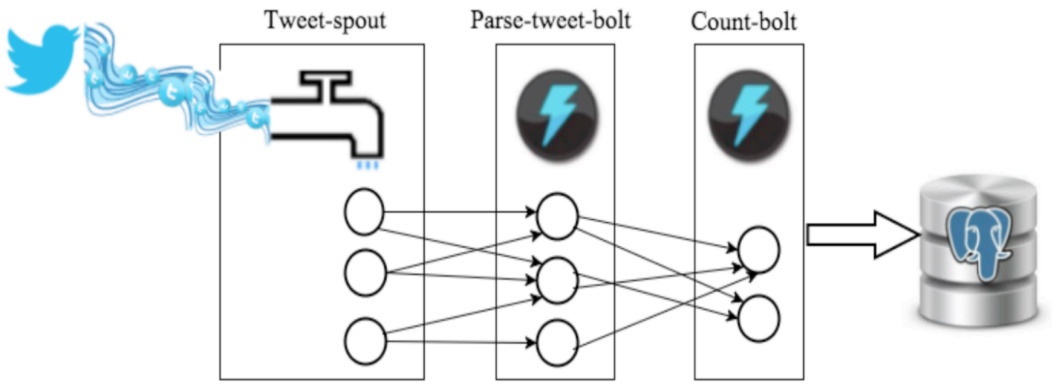
The versatility of python allows for virtually unlimited variations on the current application. For instance, the parameters can be changed to search for Tweets that only include mentions of the words "Donald" or "Trump" or "Donald Trump". One could include only Tweets of a certain length or include a certain topic. Parameters can be tuned to be as inclusive or exclusive as the end-user desires.

Architecture Description

The Twitter word count application sits on an Apache Storm foundation and builds on the versatility of the Python language to provide end-use functionality in the form of an API connection to Twitter and a PostgreSQL database. At its core, Apache Storm provides a streaming pipeline that enables real-time data processing and collection. This core functionality is accomplished through a Storm topology consisting of "spouts" and "bolts". For a detailed discussion of Storm spouts and bolts functionality it is best to consult the Apache Storm documentation.

This application makes use of three Spout instances, three "parse" bolt instances, and two "count" bolt instances. The Spout bolt pulls in a stream of Tweets as enabled through the Twitter API. The python tweepy library enables this API connection through user-provided Twitter credentials. Streamparse is a python library that enables the creation of Storm topologies in the python language. A filter is placed within the spout file that collects only those Tweets which match specific parameters. In the case of this application the filter is set to collect only those Tweets with the words [`"a", "the", "i", "you", "u"`].

See associated picture of application topology below.



Storm topologies are described as Directed Acyclic Graphs (DAGs) and so, after the twitter stream is pulled in through the spout, it is sent along the path as a series of tuples to the parse bolt. The parse bolt processes the tweets as they come through the pipeline and performs the following steps:

1. Extracts the actual tweet from the tuple.
2. Tokenizes the tweets into individual words, via a simple .split() method.
3. Undergoes a series of parsing and stripping steps which ultimately results in a series of lowercase words without any special characters or numbers.
4. The final product is appended to a list which is emitted to the next bolt along the path.

The Storm topology is set to shuffle the words, after they are processed through the parse bolt, to the different instances of the Count bolt. The Count bolt performs two primary functions:

1. It increments the local count of each word and displays the result to standard output.
2. It also inserts an object of the word into a Postgres DB if the word doesn't already exist and updates the count if it does.

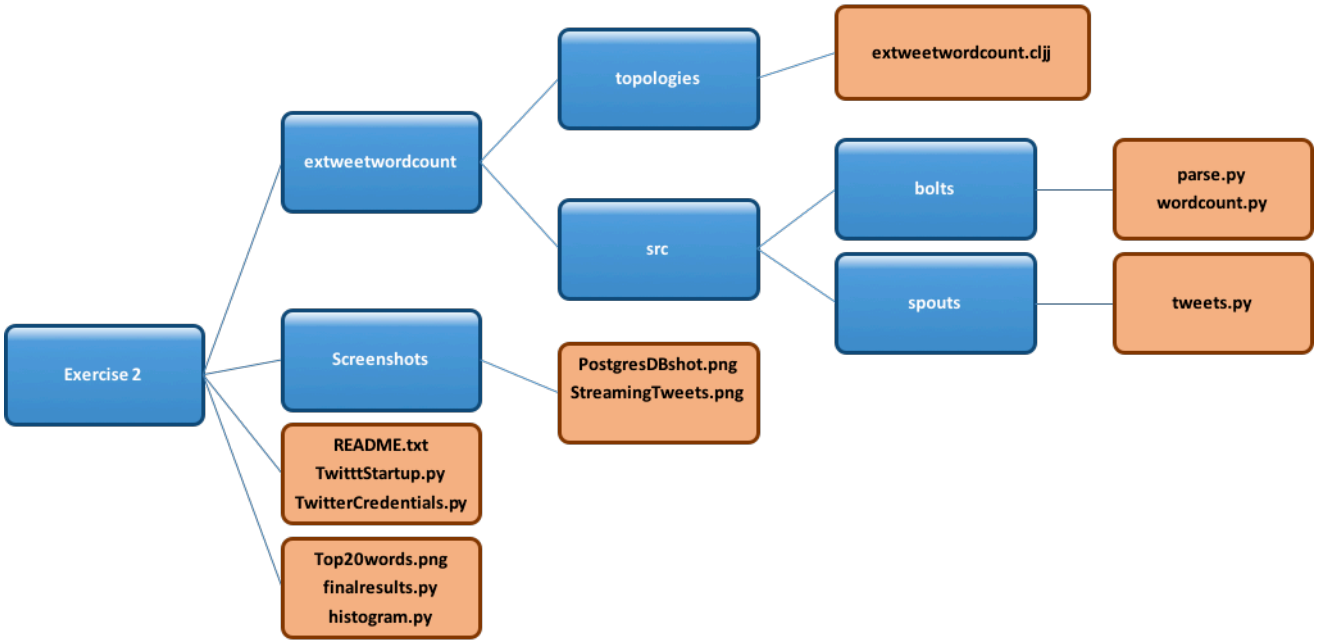
The insert and update statements in the count bolt are enabled by the python psycopg2 module which establishes a connection with the Postgres DB and then enables the execution of SQL statements.

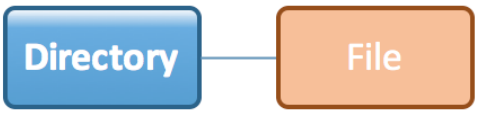
File Dependencies

The following packages/modules are required to run the Twitter application as found in this document. Modules from the python standard library are not included.

- Apache Storm
- PostgreSQL 8.6+
- Python 2.7+
    - Streamparse
    - Tweepy
    - Psycopg2

Directory of Files





Directory of Files Legend

File Descriptions

1. **README.txt:** Self-explanatory file that provides step-by-step instructions for running the application.
2. **TwitterStartup.py:** Running this file will initiate the following background steps to ensure a smoothly running process. Specifically it will:
    a. Mounts an attached EBS volume to the data directory.
    b. Starts the postgres server.
    c. Creates the tcount database and tweetwordcount table.
3. **TwitterCredentials.py:** Simple file that stores Twitter credentials, as obtained from the Twitter Developer website, as variables for later use in the application when establishing a connection with the Twitter API.
4. **Top20word.png:** Visual depiction of the top 20 words found in the Twitter stream following a half-hour run of the application.
5. **finalresults.py:** This file, when run from the command line takes in one argument (a word) from the user. If the word is found in the Postgres database the file will return the number of times the word has occurred in the stream. If the word does not exist, the file will return a "not found in the Twitter stream" message. If the user does not supply an

argument, the file will return the top 100 most occurring words in the database at that point in time.

6. **Histogram.py:** This file takes in two integers as arguments and returns all words with a total number of occurrences greater than or equal to the first integer, and less than or equal to the second integer. If the first integer entered is smaller than the second, then the script will provide a simple message telling the user to enter a smaller number first.
7. **.png files:** These files, as found in the screenshots directory, are visual depictions of the running application. Namely, the words as they are being streamed, the results of the words as stored in the database and a picture of the results of the finalresults.py and histogram.py scripts.
8. **Extweetwordcount.clj:** This file contains the code for the application topology. This file is executed from the extweetwordcount directory when the "sparse run" command is run.
9. **Spouts:**
    a. **Tweets.py:** This file is responsible for establishing a connection with the Twitter API and emitting a stream of tweets to the rest of the topology. Specifically this file accomplishes the following:
        i. Conducts authorization "handshake" with Twitter API through the use of user-provided credentials.
        ii. Creates a connection with the API and establishes a stream based on filter parameters as found in the "track" variable.
        iii. Emits the tweets in the form of tuples enabled by a Queue data structure.
10. **Bolts:**
    a. **Parse.py/wordcount.py:** Descriptions of these files can be found in the **"Architecture Description"** section of this document above.

Application Instructions

For detailed, step-by-step instructions of how to start the application, please see the accompanying README.txt file.