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, we 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 we 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.

In particular, we used one Spout and one Bolt. The Spout was based on Tweepy whose role is to connect to Twitter ad stream in related tweets for the specific ‘application’ we 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.

The bolt we used is going to parse the words from the tweets and keep a record of the accumulated counts (of the overall appearances) in the Postgres database Tcount, at tweetwodcount table.

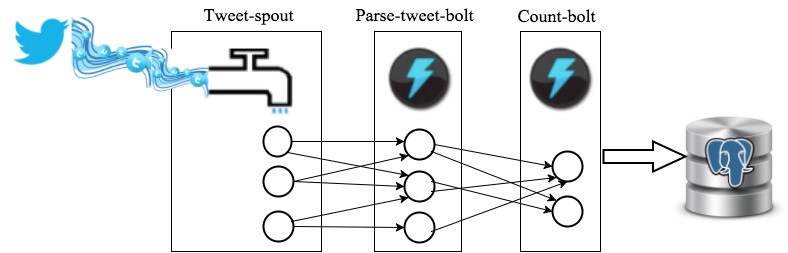


Figure 1: Application Topology

# Overall Guidelines for all steps

In this section we provide the overall guidelines for implementing the real time system as shown in Figure 1. You need to follow the below instructions for each of the steps of your implementation. You would use **ucbw205\_complete\_plus\_postgres\_PY2.7** AMI for creating your own EC2 server for this exercise. You use your github account to store your programs, data, etc.

# Step-1. Environment and Tool Setup

1. Clone the Github Repository for this exercise from git@github.com:UC-Berkeley-ISchool/w205-labs-exercises.git (If you want to just clone the exercise\_2 directory, see the appendix).

1. Create an EC2 instance using **ucbw205\_complete\_plus\_postgres\_PY2.7** and install Streamparse on it. You can use the instructions from Lab 9 to install the Streamparse. You may want to save this AMI once you configured it for the first time to save time in future re-launches. (Note: make sure python 2.7 is the default python version)

1. Install **psycopg** by running:

$ pip install psycopg2

1. Create a project called **EX2Tweetwordcount** in Streamparse.
2. Copy the files from the **tweetwordcount** directory in your cloned repository and paste them in the corresponding folders in the new **EX2Tweetwordcount** project. The description of the files in the code base is provided in Table 1.
3. Modify the **EX2tweetwordcount.clj** to implement the topology in Figure 1.

*Note1: You may not want to keep your EC2 server live when you are not working as you will run out of credit that way. So, you could save your work in github as you progress and when you launch your sever, you can re -pull the code and use.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name of the program** | | | | | | | | **Location** | **Description** | | | | |
|  | Tweets.py | |  | | | | | exercise\_2/tweetwordcount/src/spouts/ |  | Tweet-spout | |  | |
|  | Parse.py |  | | | | | | exercise\_2/tweetwordcount/src/bolts/ |  | Parse-tweet-bolt | | |  |
|  | Wordcount.py | | |  | | | | exercise\_2/tweetwordcount/src/bolts/ |  | Count-bolt |  | | |
|  | Twittercredentials.py | | | | |  | | exercise\_2/ | Twitter App Keys | | | | |
|  | hello-stream-twitter.py | | | | | |  | exercise\_2/ | Sample twitter Stream program | | | | |
|  | | | | | |
|  | tweetwordcount.clj | | | |  | | | /exercise\_2/tweetwordcount/topologies/ | Topology for the  program | | | | |
|  | | | |
| psycopg-sample.py | | | | | | | | exercise\_2/ | Sample codes on how to use psycopg | | | | |

Table 1: Description of the files in the code base

# Step-2. Twitter application setup

Once you have the topology configured with the codes in the cloned repository, you need to modify the spout program (i.e Tweets.py) to pull the tweets from twitter streaming API. In order to get the tweets, you need to set up a twitter application and get access keys for pulling the tweets out of the twitter streaming API.

The following instructions will step you through the process of acquiring data from Twitter. The first thing that you need to do is to install Tweepy, which is a python library for accessing the Twitter API.

At any point, you can revoke the access key or regenerated any of these values. To completely disable the application, you must delete the application. This does remove the consumer key, secret, and access tokens from Twitter's system and any program using them will immediately stop working.

# 2.3 Test your Application

In the code base that you cloned, hello-stream-twitter.py is a sample application that pulls tweets from the twitter streaming API. This program uses Tweepy to work with the streaming API. Use this program to test your application. Change the code in Twittercredentials.pyand insert your consumer key, consumer secret, access token, and access token secret. You should then be able to just run the program and get tweets:

$python hello-stream-twitter.py

# Step-3. Application deployment

Now you have a streamparse project and a twitter application, your task is to write codes to connect necessary pieces together as to create a full stream tweetword count processing application as depicted in Figure 1. Your application has the following pieces:

1. A spout connected to twitter streaming API that pulls the tweets from twitter stream and emits them to the parse bolt. You need to modify the necessary codes to complete this part.
2. A tweet-parse bolt that parses the tweets emitted by the spout and extracts the words out of the received tweets.
3. A tweet word count bolt that counts the number of words emitted by the tweet-parse bolt and updates the total counts for each word in a corresponding table inside a database. In order to be able to update the results in a DB, you need to create a Postgres DB called **Tcount** and a table called **Tweetwordcount** table inside **Tcount** database. You also need to modify the code in the Wordcount.pyin order to update the word count in the **Tweetwordcount** table for each word in the tweet stream. To interact with Postgres, you can use **psycopg** library**.** You can find sample codes on how to use **psycopg** in psycopg-sample.pyfile.

# Step-4. Serving Scripts

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

1. finalresults.py

This script gets a word as an argument and returns the total number of word occurrences in the stream. For example:

$ python finalresults.py hello

$ Total number of occurences of “hello”: 10

1. Running finalresults.py without an argument returns all the words in the stream and their total count of occurrences, sorted alphabetically in an ascending order, one word per line. For example:

$ python finalresults.py

$ (<word1>, 2), (<word2>, 8), (<word3>, 6), (<word4>, 1), …

1. 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. For example:

$ python histogram.py 3,8

$ <word2>: 8

<word3>: 6

<word1>: 3

**Submission Instructions:**

General guidelines:

1. All code outlined above **must be** committed and pushed to your GitHub repository.
2. All code must be runnable by your instructor in the **ucbw205\_complete\_plus\_ postgres\_PY2.7**.
3. Your GitHub repository **must be** shared with your section instructor via pull request.

Your Github repository should include:

1. Complete and fully functional twitter application codes based on the description above.
2. Architecture.pdf: a complete documentation (max 4 pages) of your twitter application including directory and file structure, application idea, description of the architecture, file dependencies, any necessary information to run the application, etc.
3. A directory called screenshots/ that has at least three screenshots of an endto-end execution of your application of your choice (name screenshots consistently, e.g.: screenshot-twitterStream.png, screenshot-stormcomponents.png, screenshot-extract-results.png).
4. Readme.txt: a file that shows the step-by-step instructions on how to run the application.
5. Plot.png: a bar chart showing the top 20 words in your twitter stream.

# Appendix

In this section we introduce some additional instructions. Some of these are from other labs and you may be able to re-use installations from those.

# Python version 2.7

If you run into issues running Streamparse, check your python version. You need to use python 2.7. Follow the instructions below, and you will need to rerun the streamparse installation.

Install the required version of Python.

$sudo yum install python27-devel –y

You can see that the Python in your execution path ($PATH) is still 2.6.X by checking the version again.

$python --version

Python 2.6.6

Rename the current version to reflect its correct version.

$mv /usr/bin/python /usr/bin/python266

Create a symbolic link from the file in the path to the version you want to execute.

$ln -s /usr/bin/python2.7 /usr/bin/python

Check that the link indeed refers to the intended version of Python.

$/usr/bin/python --version

Python 2.7.3

Check that the shell picks up the version of Python you intended.

$python --version Python 2.7.3

**Exercise 2 Grading Guidelines:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Description** | **Grading** |
| Functionality | **Complete and fully functional spout and bolts programs based on the description**:    Max 80 pt | * Correct and executable spout: 25pt * Correct and executable bolts: 30pt * Correct topology: 5pt * Finalresults.py: 10 pt * Histogram.py: 10 pt |
| Design/Architecture | **Complete documentation of the system**:    Max 20 pt | * Correct Project/Folder/ File/DB/table names: 3pt * Description/Execution instructions (readme.txt): 3pt * Architecture.pdf : 10pt * Screenshots: 2pt * Plot.png : 2 pt |