W205 Section 5 Exercise 2 Final Submission

Vincent Chu

Twitter Word Count Application

Date: 11/20/2016

datascience@berkeley

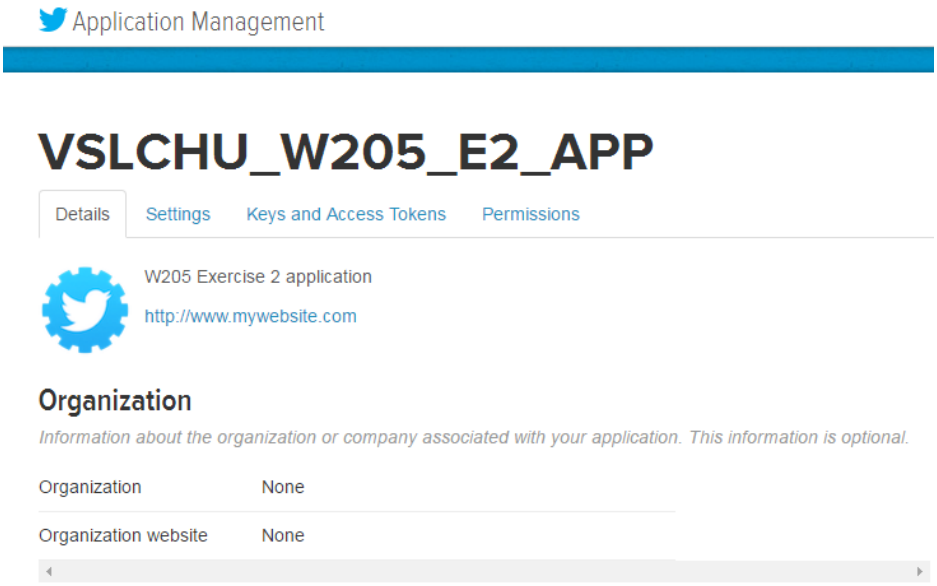
Contents

I. Introduction

Twitter has repeatedly proven to be an extremely effective communication tool for businesses, governments and even political campaigns to promote their products and/or agenda. On top of that, mining live twitter data can provide tremendous insights on close-to-real-time reactions from the social network community (i.e., what's trending). In this project, an end-to-end application was built to capture and perform word count on live tweets via the Apache Storm architecture. A serving layer with two Python scripts is designed to provide a front-end interface for end users to inquire the number of occurrence for certain key word, to get the list of key words with number of occurrences within a certain range or to examine the entire list of key words with their corresponding number of occurrences.

II. Twitter Application and Tweepy

In order to hook a Twitter stream to Apache Storm based application, a Twitter application (and account, which I never had one) must be set up.



Upon successful setup of our Twitter application, the four pieces of credential information needed to pull tweets out of the Twitter streaming API are consumer_key, consumer_secret, access_token and access_token_secret. Instead of getting access to the Twitter streaming API through traditional username/password, the OAuth open standard for authorization was used. An OAuth object generated by the OAuthHandler of the Tweepy package based on the credential information mentioned above is passed to the Twitter streaming API for authentication and authorization.

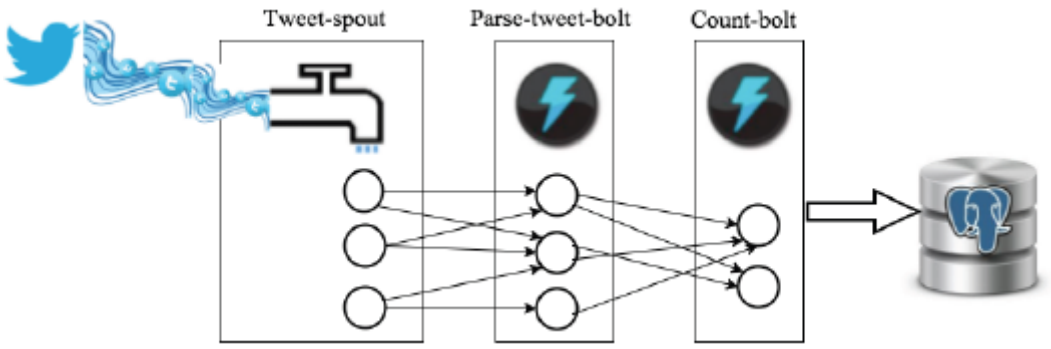
The Tweepy Python package was installed for the tweet-spout to create, listen to and get data from the live Twitter stream.

III. Apache Storm

Apache Storm is a JVM-based open source framework that supports live data streaming and is the technology chosen to implement our Tweeter Word Count Application.

i. Topology

The topology of a Storm application defines how data streams, i.e., unbounded sequence of data tuples, flow from one node to another. A topology is a direct acyclic graph (DAG) with spouts and bolts connected with grouped data streams (controlled by grouping mode such as shuffle, field, global, all, etc.). It is analogous to a Map Reduce job in the Hadoop world, except that a topology doesn't end until it is being killed explicitly. For this project, the topology file (tweetwordcount.clj) was written in Clojure, which is a Domain-specific language (DSL) to define the configuration of the spouts and the bolts. The stream of live Tweets start with the tweet-spout and then get passed to the parse-tweet-bolt and the count-bolt for processing. The parse-tweet-bolt will get stream of live Tweets from tweet-spout in a randomly shuffle grouping, whereas count-bolt will get the stream grouped by specified fields so that individual tasks of the count-bolt will be able to perform word counts correctly.



ii. Spouts

Spouts are basically sources of data and its main responsibility is to emit the stream it received to the bolts for further processing. In this project, the tweet-spout uses the Tweepy API to get authenticated and create the stream to get data from our Twitter application (VSLCHU_W205_E2_APP) and listens specifically for English tweets (using a languages=["en"] filter). It will wait for the next data tuple to come and emits the data, which will be received by the parse-tweet-bolt as configured in the topology file. An "Empty queue exception" will be reported in the log if the operation to get the next tuple from the Twitter stream timed out.

iii. Bolts

Bolts are downstream processes that processes input streams like the reduce step in MapReduce (i.e., aggregation, join, filtering, interacting with databases, etc.) and produces new streams as a result. In our project, we have two bolts working sequentially:

1. parse-tweet-bolt: Parses the input stream from tweet-spout, extract valid key words and emits them for consumption of the downstream count-bolt
2. count-bolt: Receives the stream of key valid words from parse-tweet-bolt , increment word count, and uses the psycopg2 Python package to update the count of the corresponding word in the Tweetwordcount table of the tcount database. A new row is inserted into the Tweetwordcount table if it didn't exist in the table (i.e., a new key word was found in the stream).

iv. Streamparse

Streamparse is a tool that integrates Python with Apache Storm. Its quickstart function is a handy way to set up a new streamparse project with the appropriate directory structure required to run python code on the Apache Storm infrastructure according to the topology clojure file in the topologies subdirectory. We will use this function to setup a streamparse project for our application after obtaining the source code from Github (via a bash script).

IV. Database Layer

In order to store and accumulate the number of occurrences of key words over time, a POSTGRES table named tcount was created. The tcount database contains a very simple table named Tweetwordcount with only two columns: word (text) and count(integer). Each row in this table represents a (word, number of occurrences) record. The psycopg2 Python package was installed for the count-bolt to insert new key words and update the counts as it processes the stream of words from the Twitter streaming API.

V. Serving Layer

The following two front-end scripts were developed to be serving layer for users to extract information from our Twitter Word Count application:

1. **finalresults.py**: Script that returns and prints the total number of word occurrences in the Tweeter stream (stored in POSTGRES). The user can provide a key word as an argument and the corresponding number of occurrence will be returned. Total count of all words will be returned if this argument was omitted.
2. **histogram.py**: Script to display the words with counts between the 2 numbers given as arguments by the user. If only 1 number was given, it would be treated as the lower bound. Any floating point numbers input by the user will be rounded to integers. Error messages will be displayed if any of the inputs are not numeric or if the lower bound number is larger than the upper bound number.

VI. Source Code Repository

All of the source code of this project can be obtained from the following Github repository:
https://github.com/vslchumids/vslchu_w205_ex2

i. Github Contents

File	Description	Directory
run_ex2.sh	Bash script to set up the streamparse project, create local directory structure, copy the source files into the appropriate subdirectories and start the Twitter Word Count application.	vslchu_w205_ex2/
create_tcount.sql	SQL script to create the tcount database and Tweetwordcount table	vslchu_w205_ex2/
tweetwordcount.clj	Topology clojure file	vslchu_w205_ex2/tweetwordcount/topologies/
tweets.py	Python class for tweet-spout	vslchu_w205_ex2/tweetwordcount/src/spouts/
Parse.py	Python class for parse-tweet-bolt	vslchu_w205_ex2/tweetwordcount/src/bolts /
wordcount.py	Python class for count-bolt	vslchu_w205_ex2/tweetwordcount/src/bolts/
finalresults.py	Serving script to return number of occurrences based on key word	vslchu_w205_ex2/tweetwordcount/serving/

Histogram.py	Serving script to return the list of words for which the numbers of occurrences are within a certain range	vslchu_w205_ex2/tweetwordcount/serving/
screenshot-twitterStream.png	Screenshot showing the execution of the live streaming of tweets	vslchu_w205_ex2/tweetwordcount/screenshots/
screenshot-finalresults.png	Screenshot showing results from the serving script "finalresults.py"	vslchu_w205_ex2/tweetwordcount/screenshots/
screenshot-histogram.png	Screenshot showing results from the serving script "histogram.py"	vslchu_w205_ex2/tweetwordcount/screenshots/
README.txt	Instructions on how to run this project	vslchu_w205_ex2/tweetwordcount/docs/
Architecture.pdf	Document detailing the design and architecture of this project (i.e., this document)	vslchu_w205_ex2/tweetwordcount/docs/
Plot.png	Bar chart showing the top 20 words in the Twitter stream	vslchu_w205_ex2/tweetwordcount/charts/

ii. Bash Script

The "run_ex2.sh" is perhaps the most important file of this project since it sets up the local directory structure and the streamparse project necessary to run the Twitter Word Count Application. Here are the steps it takes:

1. Create a subdirectory named exercise_2/ under the home directory for the w205 user (i.e., /home/w205)
2. Create a streamparse project under the exercise_2/ subdirectory using the "sparse quickstart" command. Remove the topology, spout and bolt files for the sample project that came from running quickstart.
3. Copy the topology clojure file, spout Python script and bolt Python scripts to the appropriate folders under the exercise_2/tweetwordcount/ subfolder for the streamparse project to run.
4. Copy the serving layer Python scripts and create POSTGRES database/table SQL script to the exercise_2 subdirectory for use by the front-end user and the bash script (*see next step below*) respectively.
5. Execute the create_tcount.sql script to create the tcount database and Tweetwordcount table in POSTGRES.
6. Run the storm based Twitter Word Count application in the exercise_2/tweetwordcount/ subdirectory.

VII. Special Note

i. Deviations from Instructions

1. Names of POSTGRES database and tables seem to be case insensitive, unless they are enclosed by double-quotes. I created the database as Tcount, however, I was able to connect to the tcount database but not the Tcount database. Therefore, I had reverted to using tcount as the name of the database in the bash script. Table names were less of an issue. All names not in quote were regarded as all lowercase, i.e., Tweetwordcount is actually an alias of tweetwordcount, which is the actual name of the table.
2. The name of the Streamparse project was renamed to tweetwordcount, since the name suggested in the assignment instructions contains a digit that that, for some reasons, seem to cause the following fatal error in Storm:
   java.lang.RuntimeException: backtype.storm.multilang.NoOutputException: Pipe to subprocess seems to be broken! No output read.