Exercise2: Tweet Word Count

Overview

Tweetwordcount is an application that counts the occurrence of words in tweets. It connects to the Twitter stream via the tweepy API, parses each tweet and increments a word counter in a PostgresSQL database. It is based on the streamparse application which uses Storm.

Directory structure

read.me.txt	Instructions on how to download and run the application
tweetwordcount	Top streamparse application folder
src	Python source files (bolts/spouts/etc.) for topologies.
bolts	A folder for all the bolts code.
__init__.py	Optional initialization. Unused.
parse.py	Tweets parser. A bolt that parsers incoming tweets and emits all the words.
wordcount.py	Persists the word count to the database. For each word the bolt receives it increments the counter for that word and updates the database. If the word does not exist, it adds a row to the tweetwordcount table with a count of one.
spouts	A folder for all the spouts code.
__init__.py	Optional initialization. Unused.
tweets.py	A spout that connect to Twitter's API using tweepy and reads all the tweets.
topologies	Contains topology definitions written using the Clojure DSL for Storm.
tweetwordcount.clj	A definition of the application topology using the closure DSL (Domain Specific Language)
virtualenvs	Contains pip requirements files in order to install dependencies on remote Storm servers.
tweetwordcount.txt	
screenshots	Application screenshots
Screenshot finalresults.png	Output of finalresults.py (linux)
Screenshot histogram.png	Output of histogram.py (linux)
Screenshot psql total couents.png	Output of psql showing number of words and rowcount (linux)
Screenshot psql while app is running2.png	Output of psql showing two queries while streamparse is running (linux)
Screenshot streamparse output (windows).png	Output of the streamparse program (taken on a windows machine)
config.json	Configuration information for all of your topologies.

finalresults.py	A python application that prints the number of mentions of a given word in Tweets. Without parameters, it prints all the words and their counts.
histogram.py	A python application that takes a range of numbers of occurrences, k1 and k2, and prints a list of words whose occurrence count was between k1 and k2 (inclusive).
fabfile.py	Optional custom fabric tasks.
project.clj	leiningen project file, can be used to add external JVM dependencies.
tasks.py	Optional code to run before and after submitting the environment definition to streamsparse. Currently empty.
Twittercredentials.py	Credetials to run twitter sample application
Histogram.pdf	Histogram of the top 20 common words found in tweets. Output from Tableau.
Architecture.pdf	This document
Plot.png	Histogram of the top 20 common words found in tweets. Output from Tableau.

Legend: **Folders**, unmodified streamparse (or UCB) files, **modified or added files**.

Database schema

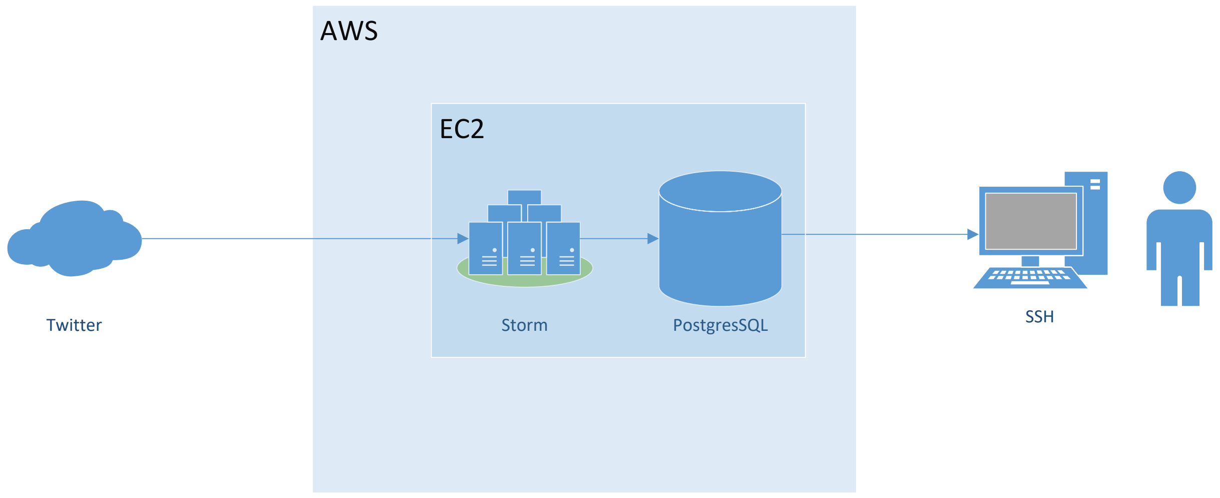
Database name: tcount

Table: tweetwordcount

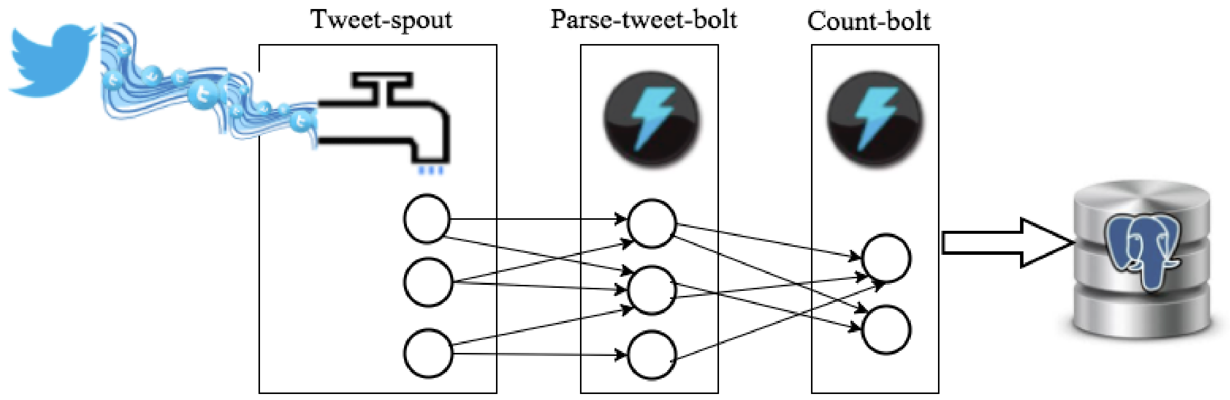
Column name	Data Type	Nullable	Key
word	TEXT	No	Primary Key
count	INTEGER	No	Index(btree)

Count is indexed to ensure good performance of histogram.py.

Application architecture



Streamparse topology



Tweet-spout (src/tweets.py)

A spout – a data source for streamparse. Connects to twitter's firehose and reads all tweets in English that contain the words 'a', 'the', 'I', 'you', or 'u'. Read timeout was adjusted to 1 second to reduce 'Empty queue exceptions'. The message is harmless but so is increasing the time the application waits for a message when to queue is empty.

Parse-tweet-bolt (src/parse.py)

A bolt. Takes a tweet as an input, parses it into words and emits each word separately. The topology is configured to spawn two such bolts (the number is a recommendation only) in shuffle mode.

Count-bolt (src/wordcount.py)

A bolt. During initialization, it reads the complete word count list from the database. This is bound (more or less) by the English dictionary, so I am not concerned about reaching any memory limits over time. It then uses this list of already seen words to determine if a word it just got is a new word that needs to be inserted, or an already existing one, whose count in the database needs to be incremented. The topology is configured to spawn two such bolts (the number is a recommendation only) in fields mode. This means that similar words will always be directed to the same bolt. This works well for our initialization technique. It ensures that the second time a word is encountered it is guaranteed to be sent to the same bolt instance, thus preventing a send insert from failing the insert on the unique key (Alternatively, we could wrap the insert with a try-except and resort to an update when an insert fails). The implemented approach is the most efficient in terms of round trips to the database since it caches the knowledge on what needs to be inserted and what updated and has no need to test for it (via select) or to guess and correct (insert with exception or update with a zero row count).