W205 Exercise 2 **March 2017**
Tom Seddon

Twitter Stream Processing Application

Application Goal

The goal of the application is to count the occurrences of words in a stream of live tweets.

The application connects to Twitter and pulls a stream of live tweets. It then parses those tweets into individual words and counts the occurrences of each word. These counts are stored in a persistent database, so that a cumulative word count can be kept across a number of different runs of the streaming component, if desired.

A serving layer allows users to see the total counts for each word, to get a count for a specific word, or to find all words that have counts within a specified range. This can be done either while the streaming process is running, or in between runs of the streaming part of the application.

The application could be used to identify twitters users interest in different brands, celebrities or sports team for example, or to monitor for words that are important for public emergencies like 'earthquake'.

Description of architecture

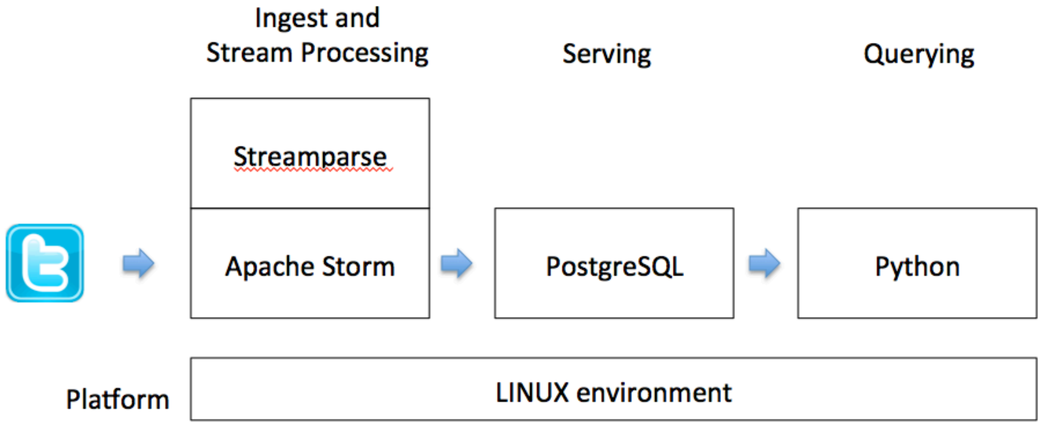
The application has a layered architecture.

The underlying platform for all components is a Linux environment (running on an Amazon m3.large EC2 instance for demonstration purposes).

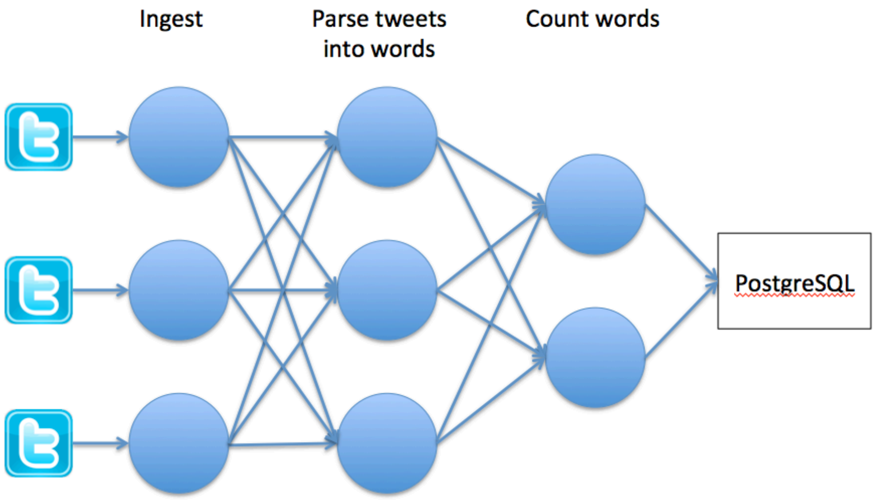
The ingestion and streaming layer is an Apache Storm stream processing topology. This is combined with the streamparse framework to allow code within the elements of the Storm topology to be written in Python, to allow the topology structure to be specified in Clojure, and to control overall invocation of the Storm processes.

The data serving layer is a PostgreSQL relational database. The final stage processes in the Storm layer write their results to this database.

The query layer is done via Python scripts, which take input from the user via command line arguments, interrogate the RDBMS and report results back to the user.



The topology implemented inside the Storm processing layer is shown below. Three parallel nodes query Twitter's live stream of tweets (using Python's Tweepy module). The tweets are distributed randomly across the next layer of three nodes that parse the individual tweets into words (and discard some common elements like 'RT'). The words are then passed into the final layer of two nodes, grouped so that the same word will always pass to the same node, so that counts can be maintained. The two nodes in this final layer both maintain their own individual count of words and increment the values in a PostgreSQL database. This topology could be scaled out with more nodes, as needed based on the application.



The PostgreSQL layer consists of a database 'Tcount', containing a single table named 'Tweetwordcount'. The 'Tweetwordcount' table has two columns:

word – text   <- Primary Key
count - integer

Directory and file structure

The files and directory structure for the application are shown below.
For clarity, the standard folders and files created by streamparse inside the extweetwordcount directory are not shown (e.g. the '_build' directory).

```
exercise_2
  |
  |- Readme.txt
  |
  |- create_Tcount.py
  |- drop_Tcount.py
  |
  |- finalresults.py
  |- histogram.py
  |
  |- extweetwordcount
      |
      +- src
      |   |
      |   +- spouts
      |   |      +- tweets.py
      |   +- bolts
      |          +- parse.py
      |          +- wordcount.py
      |
      +- topologies
      |        +- tweetwordcount.clj
      |
      + other files/folders created by streamparse, with
        no custom code added
```

<u>File Descriptions</u>

Readme.txt	instructions for how to run the application
create_Tcount.py	script to set up the PostgreSQL database and table
drop_Tcount.py	script to remove the PostgreSQL database and table
finalresults.py	script to display the results of the counts
histogram.py	script to display words within specified count range
tweets.py	code run in the ingest layer nodes of Storm topology
parse.py	code run in the parse layer nodes of Storm topology
wordcount.py	code run in the count layer nodes of Storm topology
tweetwordcount.cly	file specifying the structure of the Storm topology in Clojure

Dependencies

The application depends on the following elements:

Apache Storm
Streamparse 2.1.4
Python 2.7.3
Postgres 8.4.20
Psycopg 2.6.2
Tweepy 3.5.0
Linux

It also requires authentication keys for accessing Twitter. These keys are embedded into the tweets.py file.

The application is not dependent on one specific AMI, as long as dependencies above are met, but was developed and tested on the UC Berkeley AWS AMI:

AMI Name: UCB MIDS W205 EX2-FULL
AMI ID: ami-d4dd4ec3

How to run the application

Instructions on how to run the various elements of the application are contained in the Readme.txt file.

The application assumes that when it is started PostgreSQL is already running.

Because Twitter throttles, and can cut off, access, it is advisable to not leave the streaming component running for too long at a time -- about 30 seconds to a minute seems to be fine.