Project Architecture: Twitter application

Directory and file structure

All files for my application appear in the following GitHub repository:
https://github.com/richardhitchens/w205_exercises/tree/master/exercise_2

At the top level of this directory there are 3 files required for the project submission. In particular:

- **Architecture.pdf:** A complete documentation of my Twitter application.
- **plot.png:** A bar plot of the top 20 most frequent words from my Twitter stream.
- **readme.txt:** A file that shows the step-by-step instructions on how to run the application.

The 3 required screenshots are contained in the sub-directory **exercise_2/screenshots**. In particular:

- **screenshot-stormComponents.png:** A screen shot of my Storm topology file, which shows the three Twitter spouts, three parse bolts and two word count bolts.
- **screenshot-twitterStream.png:** A screen shot of my Twitter application in action. It in particular shows: (1) the PIDs for the three Twitter spouts, when they have an empty queue waiting on a new tweet (i.e. PIDS 22684, 22691 and 22693), and (2) the PIDS for the two word count bolts (i.e. PIDS 22687 and 22695). (Please note my parse bolt has no logging, so the stream reporting doesn't show the PIDs for the three parse bolts. They are working in the background.)
- **screenshot-topwordsResults.png:** A screen shot of my **topwords python file** call and the results from the database that were used for **plot.png**.

Within the top level directory there is also a Streamparse project in the sub-directory:
exercise_2/extweetwordcount

All additional files to those that are necessary for the Streamparse topology are contained in this sub-directory.

In particular:

- **database.py**: used to create the database **tcount** and table **tweetwordcount** in Postgres.
- **Twittercredentials.py:** which includes the keys and tokens to access my Twitter application.
- **finalresults.py:** which queries the **tweetwordcount** table to produce a word count for a specified word or all word counts if no word is specified.
- **histogram.py:** which queries the **tweetwordcount** table to produce a list of words and their counts between two user input values.
- **topwords.py:** which queries the **tweetwordcount** table and returns the top 20 most frequent words from my Twitter stream to produce **plot.png**.

The application topology is described visually in Figure 1:

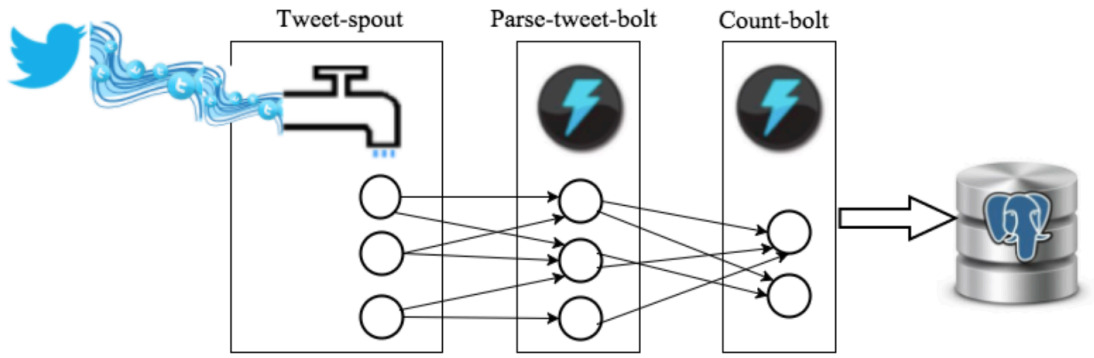

Figure 1: Application Topology

The topology files are contained in the subdirectories of exercise_2/extweetwordcount. In particular:

- **Topology:** The project topology file **tweetwordcount.clj** is in the directory exercise_2/extweetwordcount/topologies.
- **Spouts:** The project spout file **tweets.py** is in the directory exercise_2/extweetwordcount/src/spouts.
- **Bolts:** The project bolt files **parse.py** and **wordcount.py** are in the directory exercise_2/extweetwordcount/src/bolts.

Application idea

The application idea is to use the Twitter API to stream real-time tweet word counts to a Postgres database such that the database can then be queried about word counts.

Description of the architecture

My Twitter application reads the stream (using an **Apache Storm** topology written in **Clojure** as defined above) of tweets from the **Twitter streaming API** (using the **Python** package **tweepy**), it parses them, counts the occurrences of each word in the stream of tweets (using the **Python** package **streamparse**), and writes the final results back to a **Postgres** database (using the **PostgreSQL** adaptor from the **Python** package **psycopg**).

File dependencies

There are a number of file dependencies:

- The topology file (**tweetwordcount.clj)** is dependent on the spout (**tweets.py**) and bolt (**parse.py** and **wordcount.py**) files, which define the actions of the Twitter stream flow and the processing of the tweets. The spout and bolt files are dependent on the **Python** package **streamparse**, which lets you run **Python** code against real-time streams of data and integrates with **Apache Storm**.
- The spout file (**tweets.py**) is particularly dependent on the **tweepy Python** package to connect to the Twitter stream API. It is also dependent on the **Twitter credentials** of my application.
- The bolt file (wordcount.py) is in particular dependent on the **Python** package **psycopg** to connect to the **Postgres** database **tcount**.

- Each of the four **Python** scripts (**database.py**, **finalresults.py**, **histogram.py**, **topwords.py**) that interact with the **Postgres** database **tcount** are also dependent on the **Python** package **psycopg** to connect to the database.

Instructions

Create and connect to an EC2 instance using the following AMI:

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

As `root` user:

- Mount an EBS volume created as per the instructions in Lab 2 at `/data` (use `$ fdisk –l` to find the disk to be mounted), e.g. if the disk is `/dev/svdf` then run: `$ mount –t ext4 /dev/svdf /data`
- Install **psycopg** by running: `$ pip install psycopg2==2.6.2`
- Install **Tweepy** by running: `$ pip install tweepy`
- Start **Postgres** by running: `$ /data/start_postgres.sh`

Switch user to `w205` by running: `$ su - w205`

Clone my Git repository by running: `$ git clone https://github.com/richardhitchens/w205_exercises.git`

Change directory to `$ cd ~/w205_exercises/exercise_2/extweetwordcount`

Create the Postgres database and table by running: `$ python database.py`

Start the Twitter stream application: `$ sparse run`

Unlike the **hello-twitter-stream.py** example there is no time out mechanism for my application as there was no specified requirement for this in the project. After a satisfactory period of time has passed and the database has some records simply kill the stream by hitting **Ctrl + C**.

With the database now populated it is possible to run queries on the database.
To confirm the database query scripts perform their required actions please run:

- `$ python finalresults.py with`
- `$ python finalresults.py`
- `$ python histogram.py 20,30`
- `$ python topwords.py`

All done!

Shut down the instance:

- Return to `root` user: `$ exit`
- Stop Hadoop: `$ /root/stop-hadoop.sh`
- Stop Postgres: `$ /data/stop_postgres.sh`
- Terminate the EC2 instance in AWS