Twitter Stream

Directory File Structure

```
|-EXtweetwordcount
|---_build
|-----classes
|-------META-INF
|---------maven
|-----------tweetwordcount
|-------------tweetwordcount
|-----stale
|---_resources
|-----resources
|-------bolts
|-------spouts
|---logs
|---src
|-----bolts
|-----spouts
|---topologies
|---virtualenvs
```

Application Motivation

Twitter is a social media platform that allows its users to post status updates (referred to as Tweets) limited to 140 characters. This typically leads to Tweets that use less filler words and are more information dense. This, and the fact that this data is easily accessible through the Twitter API, means it's a potential gold mine of valuable data. This application is tasked with streaming the Tweets into a database that keeps track of the words used and their frequency. Exploratory analysis against this data will allow us to better understand word usage trends.

Architecture

1. **Source**: To enable the collection of Twitter data, we first need access to the source. Twitter provides developers an API that makes the tweets accessible in a standardized way. This API

provides the capability to attach to a continuous stream of Tweets. However, the status updates are only from the users that the developers account is following.

2. **Stream Processing**: We implemented a local Storm "cluster" to process each of the status updates. Storm is traditionally used with Java or Scala as the programming language, but our team uses Python everywhere else. This is why we have chosen the Python Storm implementation called Streamparse. Our Storm topology consists of one Spout to direct the incoming Tweets and two Bolts. The initial Bolt removes any characters from the Tweets we are not interested in and splits the Tweet into individual words. The second Bolt is passed the individual words from the initial Bolt and performs the Insert or Update (depending on if we have seen the word before) operation into our database.

3. **Data Storage**: We relied on a relational database called Postgres to store the tweet word counts. This decision was influenced by our need to update rows in the database as well as insert new rows. A traditional relational database is very effective at performing incremental updates and inserts. Some big data technologies, specifically map reduce technologies such as Hive, are not optimized for incremental updates and require the changes to be batched to be performant. A simple comparison showed that an insert into Hive took 19 seconds to execute (a single node on commodity hardware) and that same insert was finished in a couple hundred milliseconds in Postgres. We were treating the stream as relational data and needed to impose a level of state retention to capture the word counts. By relying on Postgres, we were able to allow it to store the state, even between application runs. If the data scales up to where the analysis becomes slow on Postgres, an additional batch step to keep Hive tables synced may be necessary.

4. **Data Analysis**: The goals of the data analysis were strictly exploratory; thus it was not necessary that a sophisticated system be put into place. Custom Python scripts were written to allow us to easily and dynamically query the database. The results of these scripts, when found interesting, were exported into Excel to create simple visualizations to better understand the word usage trends across the Tweets.

File Dependencies

It is with the assumption that anyone trying to run the twitter streaming application will do so on the UCB W205 Spring AWS EC2 AMI with Stream Parse already installed. Streamparse is the primary dependency to successfully stream the twitter word counts. In addition to Streamparse, Postgres should be started (see information below regarding README.txt instructions) and Python version 2.7 needs to be installed.

Additional Run Information

Please consult the README.txt file in the base directory of this repository. There you will find instructions on how to prepare the Postgres environment and how to successfully run this application.