Exercise 2 Architecture of Twitter App

1 HIGH LEVEL ARCHITECTURE

The application consists of several components. This section describes the spout and the bolts which can be thought of as a data source and data processors respectively.

The code for the overall topology is specified in Clojure and can be found at:

<top-level-dir>/topologies/EXtwotweetwordcount.clj

This topology file contains a spout configuration (for a single spout) and bolt configuration for two bolts.

The spout which is the source of the data is called “tweet-spout” and its output is a tweet. The spout in turn gets the tweets via tweepy from Twitter using Twitter app credentials for my app.

The output from the spout first goes to a bolt called “parse-tweet-bolt” which tokenizes the tweet into a stream of individual words. The output of this bolt then goes to another bolt called “count-bolt” which counts the number of occurrences of that word and in addition stores the words in a postgresql database table.

The code for the spout can be found at:

<top-level-dir>/src/spouts/tweets.py

The code for the “parse-tweet-bolt” can be found at:

<top-level-dir>/src/bolts/parse.py

The code for the “count-bolt” can be found at:

<top-level-dir>/src/bolts/wordcount.py

In addition there are two scripts in the top level directory:

<top-level-dir>/finalresults.py

which either gives the word count for a single specified word or the word counts of all words in the postgresql database.

The other script is:

<top-level-dir>/histogram.py

Which gives a list of word,count pairs for all words whose count falls between two specified numbers.

In addition, there are two files Twittercredentials.py and hello-stream-twitter.py that were supplied to us as part of Exercise-2. They are intended to help test the app.

The next sections describe each component of the app

2 SPOUTS AND BOLTS

The spout “tweet-spout” code is contained in tweets.py. Most of the code was supplied as sample code as part of the exercise. It consists of two classes: TweetStreamListener which acts as a listener. The other class is the Tweets class which is a sub-class of the Spout class. It instantiates a Queue, provides the Twitter app credentials to Twitter to create the stream and uses TweetStreamListener to listen for English tweets which contain certain tracking words.

The bolt “parse-tweet-bolt” code is contained in the file parse.py. It consists of a single class ParseTweet which is a subclass of the class Bolt. It receives the tweets from the spout and breaks them up into words and passes them on to the “count-bolt” bolt

The bolt “count-bolt” code is contained in wordcount.py. It contains a major portion of the deliverables for this exercise. The main class here the PGDB class which encapsulates all of the code required to store and retrieve words and their counts from the postgres database. It has the following methods:

create_db() - Creates the “tcount” database. While this method is defined it is not actually used by the app (its invocation is commented out because one of the assumptions made by this exercise is that the “tcount” database has been pre-created.

create_table() – A method that creates the “tweetwordcount” table. Again the invocation of this code is commented out as this exercise assumes that the table has been pre-created

open_db() – Opens a connection to the DB using the password and obtains a cursor.

read_table() – Method used to read records and fields from the DB table. It takes 3 arguments: the name of the table, a “fields” array of tuples which specifies the fields in a row that need to be retrieved and a ‘matcher” array of tuples which specifies how to select the rows in the table.

upsert() – The upsert method is used to either insert the word into the table if it is not already present or to update the count if the word is already present in the table. While newer versions of postgres have a native upsert method, this custom version of upsert was written to allow the app to run with older versions of postgres.

close_db() – This releases the cursor and closes the connection to the DB.

The other class in this bolt is the WordCount class which is a subclass of the Bolt class. It acts as a driver, accepting each word as it arrives from the parse bolt, updating the app’s count for that word, retrieving the stored count for that word (if any), computing the new count and then “upserting” the word and its updated count into the DB table. Note that while the app’s count only reflects the words seen in the current invocation of the app, the stored count reflects all counts over any number of invocations of the app.

3 SCRIPTS

The final two pieces of the app are the scripts “finalresults.py” and “histogram.py”

All of the code of the PGDB class (see description of “count-bolt” bolt in section 2 above) that is required for read only access is open_db(), read_table(), close_db() etc are replicated in both finalresults.py and histogram.py. While this is most certainly not good software engineering practice, making each of these scripts self-contained makes them easier to run without worrying about needing to import modules.

The finalresults.py script has two functions (in addition to the PGDB class), one to retrieve all words and their counts from the database (called print_all()) and another called print_one() to print word count for a single word, passed as argument to the script.

The histogram.py script has in addition to the PGDB class, a function called print_interval, that retrieves all the words from the table and then filters out words whose count does not lie between two specified limits (passed as an argument to the script). The “main” part of this script, parses the interval specification and makes sure that the values supplied are sane (such as the limits being integers, the lower limit being <= the upper limit, etc.

4 CAVEATS

While I provide the create_db() and create_table() methods in the PGDB class for completeness and for my testing, they have not been extensively tested and are therefore not actually invoked by my app (their invocation is commented out.). These are to be considered as beta quality aids to allow me to test the other parts of the app and are not part of the deliverables for this project.