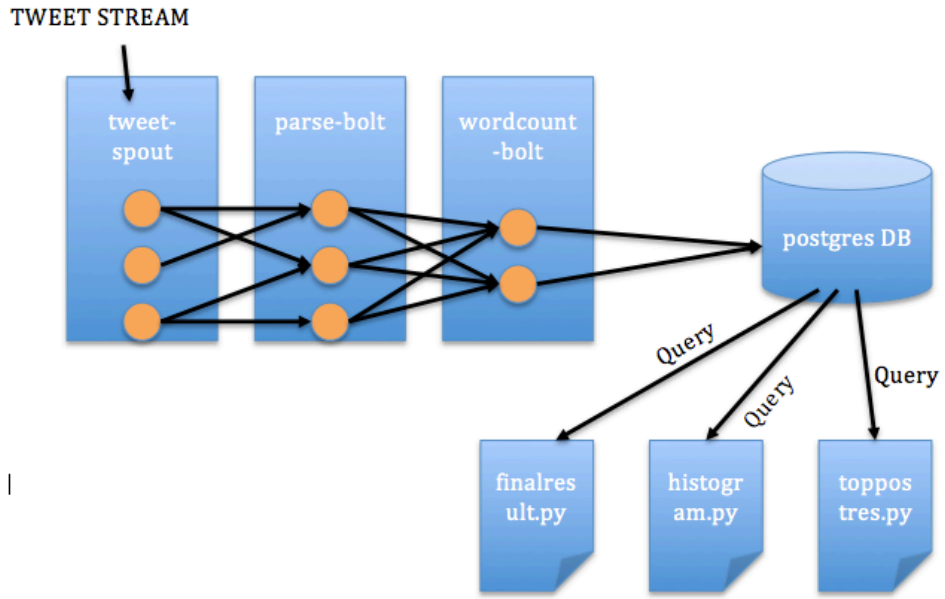
Architecture of the twitter application

1. Description of the architecture
   a. The twitter application uses the following topology. Tweeter stream is received by tweet spout and passed to two bolts for parsing, and counting respectively. Spout uses 3 processes, parsing bolts uses 3 proecess and wordcount bolt uses 2 processes. This is represented accordingly on topology.
   b. The resulting lists of words & its counts are saved into postgres DB.

<Figure 1: Project Topology>



2. Directory and file structure
   a. tweetword.clj

```
(ns tweetwordcount
  (:use     [streamparse.specs])
  (:gen-class))

# setting up a topology definition, also named <filename>.
(defn tweetwordcount [options]
   [
    # spout configuration
    {"tweet-spout" (python-spout-spec
          # topology options passed in
          options
          # python class to run, spouts.<filename>.<class>
          "spouts.tweets.Tweets"
          # output specification, what named fields will this spout emit?
          ["tweet"]
          :p 3
          )
    }
    # bolt configuration
    {"parse-tweet-bolt" (python-bolt-spec
          options
```

```
        # inputs, where does this bolt receive is tuples from?
        {"tweet-spout" :shuffle}
        # python class to run, bolts.<filename>.<class>
        "bolts.parse.ParseTweet"
        # output specification, what named fields will this spout emit?
        ["word"]
        :p 3
        )
    "count-bolt" (python-bolt-spec
        options
        # inputs, where does this bolt receive its tuples from?
        {"parse-tweet-bolt" ["word"]}
        # python class to run, bolts.<filename>.<class>
        "bolts.wordcount.WordCounter"
        # output specification, what named fields will this spout emit?
        ["word" "count"]
        :p 2
        )
  }
 ]
)
```

b. tweets.py
This spout gets twitter streaming using my tweeter credential. I inserted my twitter authentication codes. Please refer to github to view code.

c. parse.py
Parse bolts parses incoming stream into individual words. Please refer to github to view code.

d. wordcount.py
Wordcount bolts counts words and saves the result to postgres DB. The code is written that it continually **accumulate** word counts **to the same** database every time twitter application is run.

```
from __future__ import absolute_import, print_function, unicode_literals
from collections import Counter
from streamparse.bolt import Bolt
import psycopg2
import sys
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT


class WordCounter(Bolt):

    def initialize(self, conf, ctx):
        self.counts = Counter()

         # connect to tcount database
        conn = psycopg2.connect(database="tcount", user="postgres", password="pass",
host="localhost", port="5432")
        cur = conn.cursor()

        tableName = 'tweetwordcount'
        cur.execute("select * from information_schema.tables where table_name=%s",
(tableName,))

        if not cur.rowcount:
            cur.execute('''CREATE TABLE tweetwordcount
                (word TEXT PRIMARY KEY     NOT NULL,
                count INT     NOT NULL);''')

        conn.commit()
        conn.close()

    def process(self, tup):
```

```
        # Change the word into lowercase

        word = tup.values[0]
        lowerWord = word.lower()

        # Increment the local count

        self.counts[lowerWord] += 1

        isExists = self.checkRowPostGres(lowerWord)

        if not isExists:
            self.insertPostGres(lowerWord)
        else:
            self.updatePostGres(lowerWord)

        self.emit([lowerWord, self.counts[lowerWord]])

        # Log the count - just to see the topology running
        self.log('%s: %d' % (lowerWord, self.counts[lowerWord]))

    def insertPostGres(self, word):

        conn = psycopg2.connect(database="tcount", user="postgres", password="pass",
host="localhost", port="5432")

        cur = conn.cursor()

        initCount = 1

        # Insert into tcount using word, count.
        cur.execute("INSERT INTO tweetwordcount (word,count) \
            VALUES (%s, %s)", (word, initCount));
        conn.commit()
        conn.close()

    def updatePostGres(self, word):

        conn = psycopg2.connect(database="tcount", user="postgres", password="pass",
host="localhost", port="5432")

        cur = conn.cursor()

        #Update

        cur.execute("UPDATE tweetwordcount SET count = count + 1 WHERE word=%s",
(word,))
        conn.commit()
        conn.close()

         # Check if the word already exists in postgres in order to avoid primary key
         issue

    def checkRowPostGres(self, word):

        conn = psycopg2.connect(database="tcount", user="postgres", password="pass",
host="localhost", port="5432")

        cur = conn.cursor()
        cur.execute("SELECT * FROM tweetwordcount WHERE word=%s", (word,))
        check = cur.fetchall()

        isExists = False
        if check:
            isExists = True

        conn.commit()
        conn.close()

        return isExists
```

e. finalresults.py
This python script queries from the database for all the words inside it and its corresponding counts. If there is no record, it prints "Empty Data". If there is a record, it fetches all data and prints the words in alphabetically ascending order.

If you give it an argument, a *word*, it will only return that word and its count. For example, if you wanted to search for a word, 'christmas', it queries the database for the word and returns its counts. Syntax for giving the argument is as follows:

```
python filanresults.py
python finalresults.py <word>
```

f. histogram.py
This python script queries from the database for the words within pre-specified occurrence number ranges. For example, it gives all the words that occurred between k1, and k2. Syntax for giving the argument is as follows:

```
python histogram.py k1, k2
```

g. toppostgres.py
This python script queries from the database for words with top 20 occurrences. Syntax for running this script is as follows:

```
python toppostgres.py
```

3. File dependencies
   a. For twitter application, you need the following files under appropriate folder
      i. EXTweet.wordcount.clj
      `/EX2Tweetwordcount/topologies`

      ii. tweets.py
      `/EX2Tweetwordcount/src/spouts`

      iii. parse.py
      `/EX2Tweetwordcount/src/bolts`

      iv. wordcount.py
      `/EX2Tweetwordcount/src/bolts`

      v. createdatabase.py: This python script creates database in postgres. **You must run this python script before running twitter application**. wordcount.py requires an active database already installed inside postgres. So withouth running this script first, 'sparse run' will not execute properly.
      `/EX2Tweetwordcount/`

   b. For queries, you need the following file under appropriate folder
      i. finalresults.py
      `/EX2Tweetwordcount/`
      ii. histogram.py
      `/EX2Tweetwordcount/`
      iii. topposgres.py
      `/EX2Tweetwordcount/`