Tweetcounter Application

Adam Lenart

November 10, 2016

1 Application idea and description

The tweetcounter application aims to count the frequency of unigram expressions that appear in tweets. With the help of Streamparse, a real-time stream of tweets are parsed in Apache Storm using Python. The main components of the Storm architecture are the spouts and bolts and the topology which describes their numbers and behavior. In this application, the spout is the source of information which channels the Twitter stream to the bolts to process it. In the topology (Figure 1) of this tweetcounter application, there are two bolts: a bolt that parses the tweets and a bolt that counts the occurrence of the words in the tweets. The parser bolt splits parses unigrams by splitting the tweets on a whitespace character and filtering out hash tags, user mentions, retweet tags, urls and non-ASCII strings. The wordcounter bolt counts the parsed unigrams locally and registers them in a PostgreSQL database. This application contains two topologies, one that opens a new database each time the application is called, and another one which accumulates the parsed tweets in an existing database.

Once the unigrams are registered in the database, Python serving scripts allow to query relevant information from them such as the frequency of an expression, listing all of the expressions within a limit of frequencies or simply listing all of the expressions and their counts.

All computations are performed on an Amazon AWS EC2 instance.

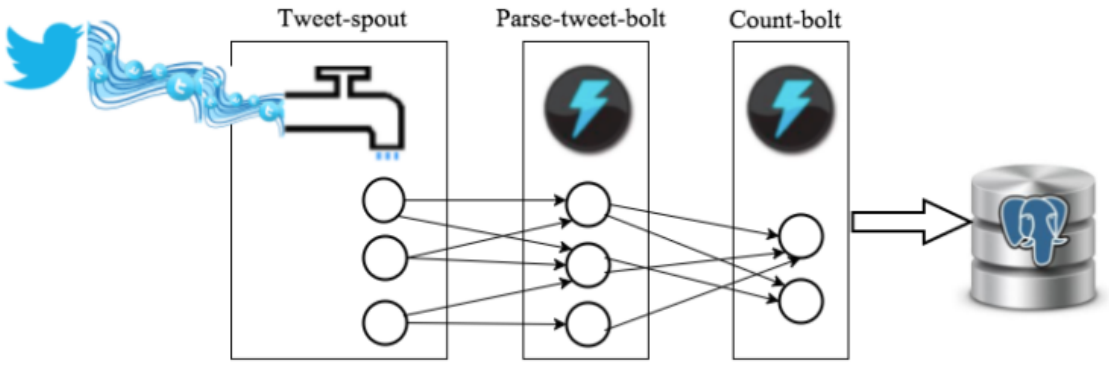

Figure 1: Application topology. Image taken from UCB MIDS W205 Fall 16 Exercise 2 description.

2 Directory and file structure

Name	Location	Description
Architecture.pdf	exercise_2	Description of application
EX2Tweetwrodcount/	exercise_2	Tweetwordcount application
config.json	EX2Tweetwordcount/	App configuration
fabfile.py	EX2Tweetwordcount/	Custom fabric tasks
logfile	EX2Tweetwordcount/	Optional logs
project.clj	EX2Tweetwordcount/	Leiningen project file
tasks.py	EX2Tweetwordcount/	Custom invoke tasks
src/	EX2Tweetwordcount/	Python source files
bolts/	src/	Python source files of the bolts
parse.py	bolts/	Parse bolt source file
wordcount.py	bolts/	Wordcount bolt source file
wordcount_cumulative.py	bolts/	Cumulative wordcount bolt source file
spouts/	EX2Tweetwordcount	Python source files of the spouts
tweets.py	spouts/	Tweet spout source file
topologies/	EX2Tweetwordcount/	Topology definitions
tweetwordcount.clj	topologies/	Wordcount topology Clojure file
tweetwrodcount_cumulative.clj	topologies/	Cumulative wordcount topology Clojure file
virtualenvs/	EX2Tweetwordcount/	Pip requirements for remote Storm servers
wordtext.txt	virtualenvs/	List of dependencies to install on remote Storm servers
misc/	exercise_2/	Miscellaneous files
plot_bar.py	misc/	Python source for bar plot
Plot.png	misc/	Bar plot of 20 most frequent words
wordfreq.txt	misc/	List of the most frequent English words
screenshots/	exercise_2	Contains screenshots
serving-scripts/	exercise_2	Serving Python scripts
finalresults.py	serving-scripts/	Extracts unigram counts
histogram.py	serving-scripts/	Lists unigram counts in frequency interval

Table 1: Directory and file structure

Additionally, the source folders contain __init__.py files which allow their usage as Python modules and readme.md files that provide a description of the folders. Please see the appendix for a graphical overview of the directory structure (Figure 2).

3 Usage

First, install the `tweepy` and `psycopg2` Python libraries and start a PostrgreSQL server by

```
pg_ctl -D /home/w205/data -l logfile start
```

Then enter `EX2Tweetwordcount` and start the application either by calling the `tweetwordcount` or `tweetwordcount_cumulative` topologies for an application that starts with a new (and drops an existing) table or an application that adds to an existing table, respectively.

```
sparse run -n tweetwordcount
```

or

```
sparse run -n tweetwordcount_cumulative
```

The `serving-scripts` folder contains convenience Python 2 scripts to query the PostgreSQL database. Type

```
python histogram.py lower,upper
```

for a `lower` and an `upper` positive integer to print out all of the unigrams whose counts are in the [`lower,upper`] interval.
The `finalresults.py` script can be called either with or without an argument such as

```
python finalresults.py
```

or

```
python finalresults.py string
```

to print out the list of word and count tuples or the frequency of the queried `string`.

4 Appendix

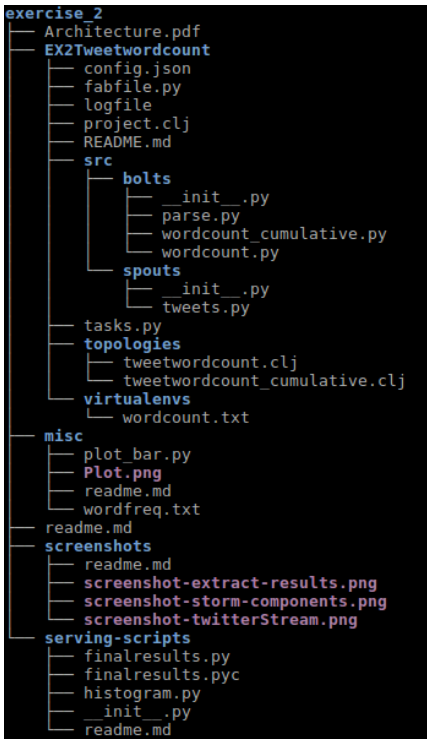

Figure 2: Directory structure